



System i

# Object signing and signature verification

*Version 5 Release 4*







System i

Object signing and signature verification

*Version 5 Release 4*

**Note**

Before using this information and the product it supports, read the information in “Notices,” on page 45.

**Fourth Edition (February 2006)**

This edition applies to version 5, release 4, modification 0 of IBM i5/OS (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 2002, 2006. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

## Object signing and signature verification 1

Printable PDF . . . . .	1
Object signing concepts . . . . .	2
Digital signatures . . . . .	2
Signable objects . . . . .	3
Object signing processing . . . . .	4
Signature verification processing . . . . .	5
Code checker integrity verification function . . . . .	5
Object signing scenarios . . . . .	6
Scenario: Using DCM to sign objects and verify signatures . . . . .	6
Scenario: Using APIs to sign objects and verify object signatures . . . . .	15
Scenario: Using iSeries Navigator Management Central to sign objects . . . . .	25
Object signing and signature verification prerequisites . . . . .	33
Managing signed objects . . . . .	35

System values and commands that affect signed objects . . . . .	35
Save and restore considerations for signed objects	38
Code checker commands to ensure signature integrity . . . . .	39
Verifying code checker function integrity . . . . .	41
Troubleshooting signed objects . . . . .	41
Troubleshooting object signing errors . . . . .	41
Troubleshooting signature verification errors . . . . .	42
Interpreting code checker verification error messages . . . . .	42
Related information for object signing and signature verification . . . . .	44

## Appendix. Notices . . . . . 45

Trademarks . . . . .	47
Terms and conditions . . . . .	47



---

## Object signing and signature verification

Find information about i5/OS<sup>®</sup> object signing and signature verification security capabilities that you can use to ensure the integrity of objects. Learn how to use one of several i5/OS methods for creating digital signatures on objects to identify the source of the object and provide a means for detecting changes to the object. Also learn how to enhance system security by verifying digital signatures on objects, including operating system objects, to determine whether there have been changes to the contents of the object since it was signed.

Object signing and signature verification are security capabilities that you can employ to verify the integrity of a variety of objects. You use a digital certificate's private key to sign an object, and you use the certificate (which contains the corresponding public key) to verify the digital signature. A digital signature ensures the integrity of time and content of the object that you are signing. The signature provides proof of both authenticity and authorization. It can be used to show proof of origin and detect tampering. By signing the object, you identify the source of the object and provide a means for detecting changes to the object. When you verify the signature on an object you can determine whether there have been changes to the contents of the object since it was signed. You can also verify the source of the signature to ensure the reliability of the object's origin.

You can implement object signing and signature verification by:

- APIs to sign objects and to verify the signatures on objects programmatically.
- Digital Certificate Manager to sign objects and to view or to verify object signatures.
- iSeries<sup>™</sup> Navigator Management Central to sign objects as part of distributing packages for other systems to use.
- CL commands, such as Check Object Integrity (CHKOBJITG) to verify signatures.

To learn more about these methods of signing objects and how signing objects can enhance your current security policy, review these topics:

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 44.

---

### Printable PDF

Use this information to print the entire topic of i5/OS object signing and signature verification as a PDF file.

To view or download the PDF version of this document, select Object signing and signature verification (file size 605 KB).

#### Saving PDF files:

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser (right-click the link above).
2. Click **Save Target As...** if you are using Internet Explorer. Click **Save Link As...** if you are using Netscape Communicator.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

#### Downloading Adobe Acrobat Reader

You need Adobe Acrobat Reader to view or print these PDFs. You can download a copy from the Adobe Web site ([www.adobe.com/products/acrobat/readstep.html](http://www.adobe.com/products/acrobat/readstep.html)) .

---

## Object signing concepts

This topic provides concept and reference information about i5/OS digital signatures and how the i5/OS object signing and signature verification processes work.

Before you begin using object signing and signature verification capabilities, you may find it helpful to review some of these concepts:

### Digital signatures

This topic provides information about what i5/OS digital signatures are and what protection they provide.

i5/OS provides support for using digital certificates to digitally "sign" objects. A digital signature on an object is created by using a form of cryptography and is like a personal signature on a written document. A digital signature provides proof of the object's origin and a means by which to verify the object's integrity. A digital certificate owner "signs" an object by using the certificate's private key. The recipient of the object uses the certificate's corresponding public key to decrypt the signature, which verifies the integrity of the signed object and verifies the sender as the source.

Object signing support augments traditional system tools for controlling who can change objects. Traditional controls cannot protect an object from unauthorized tampering while the object is in transit across the Internet or other untrusted network. Because you can detect whether the contents of an object have been changed since they were signed, you can more easily determine whether to trust objects that you obtain in cases such as these.

A digital signature is an encrypted mathematical summary of the data in an object. The object and its contents are not encrypted and made private by the digital signature; however, the summary itself is encrypted to prevent unauthorized changes to it. Anyone who wants to ensure that the object has not been changed in transit and that the object originated from an accepted, legitimate source can use the signing certificate's public key to verify the original digital signature. If the signature no longer matches, the data may have been altered. In such a case, the recipient can avoid using the object and can instead contact the signer to obtain another copy of the signed object.

The signature on an object represents the system that signed the object, not a specific user on that system (although the user must have the appropriate authority to use the certificate for signing objects).

If you decide that using digital signatures fits your security needs and policies, you need to evaluate whether to use public certificates versus issuing local certificates. If you intend to distribute objects to users in the general public, you need to consider using certificates from a well-known public Certificate Authority (CA) to sign objects. Using public certificates ensures that others can easily and inexpensively verify the signatures that you place on objects that you distribute to them. If, however, you intend to distribute objects solely within your organization, you may prefer to use Digital Certificate Manager (DCM) to operate your own Local CA to issue certificates for signing objects. Using private certificates from a Local CA to sign objects is less expensive than purchasing certificates from a well-known public CA.

### Types of digital signatures

Beginning in V5R2, you can sign command (\*CMD) objects; you also can choose one of two types of signatures for \*CMD objects: core object signatures or entire object signatures.

- **Entire object signatures** This type of signature includes all but a few nonessential bytes of the object.

- **Core object signatures** This type of signature includes the essential bytes of the \*CMD object. However, the signature does not include those bytes that are subject to more frequent changes. This type of signature allows some changes to be made to the command without invalidating the signature. Which bytes the core object signature does not include vary based on the specific \*CMD object; core signatures do not include parameter defaults on the \*CMD objects, for example. Examples of changes that will not invalidate a core object signature include:
  - Changing command defaults.
  - Adding a validity checking program to a command that does not have one.
  - Changing the Where allowed to run parameter.
  - Changing the Allow limited users parameter.

#### **Related concepts**

“Signable objects”

This topic provides information about which objects you can sign and about i5/OS command (\*CMD) object signature options.

#### **Related information**

Digital Certificate Manager (DCM)

## **Signable objects**

This topic provides information about which objects you can sign and about i5/OS command (\*CMD) object signature options.

You can digitally sign a variety of i5/OS object types, regardless of the method that you use to sign them. You can sign any object (\*STMF) that you store in the system’s integrated file system, except objects that are stored in a library. If the object has an attached Java™ program, the program will also be signed. You can sign only these objects in the QSYS.LIB file system: programs (\*PGM), service programs (\*SRVPGM), modules (\*MODULE), SQL packages (\*SQLPKG), \*FILE (save file only), and commands (\*CMD).

To sign an object, it must reside on the local system. For example, if you operate a Windows® 2000 server on an Integrated xSeries® Server for System i™, you have the QNTC file system available in the integrated file system. The directories in this file system are not considered local because they contain files that are owned by the Windows 2000 operating system. Also, you cannot sign empty objects or objects that are compiled for a release before V5R1.

## **Command (\*CMD) object signatures**

When you sign \*CMD objects, you can choose one of two types of digital signatures to apply to the \*CMD object. You can elect either to sign the entire object, or to sign the core part of the object only. When you elect to sign the entire object, the signature is applied to all but a few nonessential bytes of the object. The entire object signature includes the items contained in the core object signature.

When you elect to sign only the core object, the essential bytes are protected by the signature while bytes that are subject to more frequent changes are not signed. Which bytes are unsigned varies based on the \*CMD object, but can include bytes that determine the mode in which the object is valid or determine where the object is allowed to run, among others. Core signatures do not include parameter defaults on the \*CMD objects, for example. This type of signature allows some changes to be made to the command without invalidating its signature. Examples of changes that will not invalidate these types of signatures include:

- Changing command defaults.
- Adding a validity checking program to a command that does not have one.
- Changing the Where allowed to run’ parameter.
- Changing the Allow limited users parameter.

The following table describes exactly which bytes in a \*CMD object are included as part of the core object signature.

### Composition of core object signature on \*CMD objects

Part of object	Relationship to core object signature
Command defaults changed by CHGCMDDFT	Not part of the core object signature
Program to process command and library	Always included as part of the core object signature
REXX source file and library	Included if specified for the command at the time of signing, otherwise not part of the core object signature
REXX source member	Included if specified for the command at the time of signing, otherwise not part of the core object signature
REXX command environment and library	Included if specified for the command at the time of signing, otherwise not part of the core object signature
REXX exit program name, library, and exit code	Included if specified for the command at the time of signing, otherwise not part of the core object signature
Validity checking program and library	Included if specified for the command at the time of signing, otherwise not part of the core object signature
Mode in which valid	Not part of the core object signature
Where allowed to run	Not part of the core object signature
Allow limited users	Not part of the core object signature
Help bookshelf	Included if specified for the command at the time of signing, otherwise not part of the core object signature
Help panel group and library	Included if specified for the command at the time of signing, otherwise not part of the core object signature
Help identifier	Included if specified for the command at the time of signing, otherwise not part of the core object signature
Help search index and library	Included if specified for the command at the time of signing, otherwise not part of the core object signature
Current library	Included if specified for the command at the time of signing, otherwise not part of the core object signature
Product library	Included if specified for the command at the time of signing, otherwise not part of the core object signature
Prompt override program and library	Included if specified for the command at the time of signing, otherwise not part of the core object signature
Text (description)	Not part of either a core object signature or an entire object signature because it is not stored in the object
Enable graphical user interface (GUI)	Not part of the core object signature

#### Related concepts

“Digital signatures” on page 2

This topic provides information about what i5/OS digital signatures are and what protection they provide.

### Object signing processing

This topic provides information about how the process of signing objects on your system running the i5/OS operating system works and what parameters you can set for the process.

When you sign objects you can specify the following options for object signing processing.

### **Error processing**

You can specify what type of error processing the application is to use when creating signatures on more than one object. You can specify that the application either stop signing objects when an error occurs or continue signing any other objects in the process.

### **Duplicate object signature**

You can specify how the application is to handle the signing process when the application is re-signing an object. You can specify whether to leave the original signature in place or to replace the original signature with the new signature.

### **Objects in subdirectories**

You can specify how the application is to handle signing objects in subdirectories. You can specify that the application individually sign objects in any subdirectories or that the application only sign those objects within the main directory while ignoring all subdirectories.

### **Scope of object signature**

When signing \*CMD objects, you can specify whether to sign the entire object or to sign the core part of the object only.

## **Signature verification processing**

Learn how the i5/OS process of verifying an object signature works and what parameters you can set for the process.

You can specify the following options for signature verification processing.

### **Error processing**

You can specify what type of error processing the application is to use when verifying signatures on more than one object. You can specify that the application either stop verifying signatures when an error occurs or continue verifying signatures on any other objects in the process.

### **Objects in subdirectories**

You can specify how the application is to handle verifying signatures on objects in subdirectories. You can specify that the application individually verify signatures on objects in any subdirectories or that the application only verify signatures for those objects within the main directory while ignoring all subdirectories.

### **Core versus entire signature verification**

There are system rules that determine how the system is to handle core and entire signatures on objects during the verification process. These rules are as follows:

- If there are no signatures on the object, the verify process reports the object is not signed and continues verifying any other objects in the process.
- If the object was signed by a system trusted source (IBM®), the signature must match or the verification process fails. If the signature matches, the verification process continues. The signature is an encrypted mathematical summary of the data in the object; therefore, the signature is considered to match if the data in the object during verification matches the data in the object when it was signed.
- If the object has any entire object signatures that are trusted (based on certificates contained in the \*SIGNATUREVERIFICATION certificate store), at least one of these signatures must match or the verification process fails. If at least one entire object signature matches, the verification process continues.
- If the object has any core object signatures that are trusted, at least one of these must match a certificate in the \*SIGNATUREVERIFICATION certificate store or the verify process fails. If at least one core object signature matches the verification process continues.

## **Code checker integrity verification function**

This topic provides information about how you can verify the integrity of the code checker function that you use to verify the integrity of your system running the i5/OS operating system.

Beginning in V5R2, i5/OS shipped with a code checking function that you can use to verify the integrity of signed objects on your system, including all operating system code that IBM ships and signs for your system. Now in V5R3, you can use a new Application Programming Interface (API) to verify the integrity of the code checking function itself, as well as key operating system objects.

The Check System (QydoCheckSystem) API provides i5/OS system integrity verification. You use this API to verify the programs (\*PGM) and service programs (\*SRVPGM) and selected command (\*CMD) objects in the QSYS library. Additionally, the Check System API tests the Restore object (RSTOBJ) command, the Restore Library (RSTLIB) command, the Check Object Integrity (CHKOBJITG) command, and Verify Object API. This test ensures that these commands and the Verify Object API report signature validation errors when appropriate; for example, when a system supplied object is not signed or contains an invalid signature.

The Check System API reports error messages for verification failures and other errors or verification failures to the job log. However, you can also specify one of two additional error reporting methods, depending on how you set the following options:

- If the QAUDLVL system value is set to \*AUDFAIL, then the Check System API generates auditing records to report any failures and errors that the Restore Object (RSTOBJ), Restore Library (RSTLIB), and Check Object Integrity (CHKOBJITG) commands find.
- If the user specifies that the Check System API use a results file in the integrated file system, then the API either creates the file if it does not exist or the API appends to the file to report any errors or failures that the API finds.

#### **Related tasks**

“Verifying code checker function integrity” on page 41

Learn how to verify the integrity of the code checker function that you use to verify i5/OS system integrity.

---

## **Object signing scenarios**

Review the scenarios that illustrate some typical situations for using i5/OS object signing and signature verification capabilities. Each scenario also provides the configuration tasks you must perform to implement the scenario as described.

Your system provides several different methods for signing objects and verifying signatures on objects. How you choose to sign objects and how you work with signed objects varies based on your business and security needs and objectives. In some cases, you may need only to verify object signatures on your system to ensure that object integrity is intact. In other cases, you may choose to sign objects that you distribute to others. Signing the objects allows others to identify the origin of the objects and to check the integrity of the objects.

Which method you choose to use depends on a variety of factors. The scenarios provided in this topic describe some of the more common object signing and signature verification objectives within typical business contexts. Each scenario also describes any prerequisites and the tasks that you must perform to implement the scenario as described. Review these scenarios to help you determine how you can use object signing capabilities in a way that best suits your business and security needs:

### **Scenario: Using DCM to sign objects and verify signatures**

This scenario describes a company that wants to sign vulnerable application objects on their public Web server. They want to be able to more easily determine when there are unauthorized changes to these objects. Based on the company’s business needs and security goals, this scenario describes how to use Digital Certificate Manager (DCM) as the primary method for using i5/OS object signing capabilities.

## Situation

As an administrator for MyCo, Inc. you are responsible for managing your company's two systems. One of these systems provides a public Web site for your company. You use the company's internal production system to develop the content for this public Web site and transfer the files and program objects to the public Web server after testing them.

The company's public Web server provides a general company information Web site. The Web site also provides various forms that customers fill out to register products, and to request product information, product update notices, product distribution locations, and so forth. You are concerned about the vulnerability of the cgi-bin programs that provide these forms; you know that they might be altered. Therefore, you want to be able to check the integrity of these program objects and to detect when unauthorized changes have been made to them. Consequently, you have decided to digitally sign these objects to accomplish this security goal.

You have researched i5/OS object signing capabilities and have learned that there are several methods that you can use to sign objects and verify object signatures. Because you are responsible for managing a small number of systems and do not feel that you will need to sign objects often, you have decided to use Digital Certificate Manager (DCM) for performing these tasks. You have also decided to create a Local Certificate Authority (CA) and use a private certificate to sign objects. Using a private certificate issued by a Local CA for object signing limits the expense of using this security technology because you do not have to purchase a certificate from a well-known public CA.

This example serves as a useful introduction to the steps involved in setting up and using object signing when you want to sign objects on a small number of systems.

## Scenario advantages

This scenario has the following advantages:

- Signing objects provides you with a means to check the integrity of vulnerable objects and more easily determine whether objects have been changed after they have been signed. This may reduce some of the troubleshooting that you do in the future to track down application and other system problems.
- Using DCM's graphical user interface (GUI) to sign objects and verify object signatures allows you and others in the company to perform these tasks quickly and easily.
- Using DCM to sign objects and verify object signatures reduces the amount of time you must spend to understand and use object signing as part of your security strategy.
- Using a certificate issued by a Local Certificate Authority (CA) to sign objects makes signing objects less expensive to implement.

## Objectives

In this scenario, you want to digitally sign vulnerable objects, such as cgi-bin programs that generate forms, on your company's public server. As the system administrator at MyCo, Inc, you want to use Digital Certificate Manager (DCM) to sign these objects and to verify the signatures on the objects.

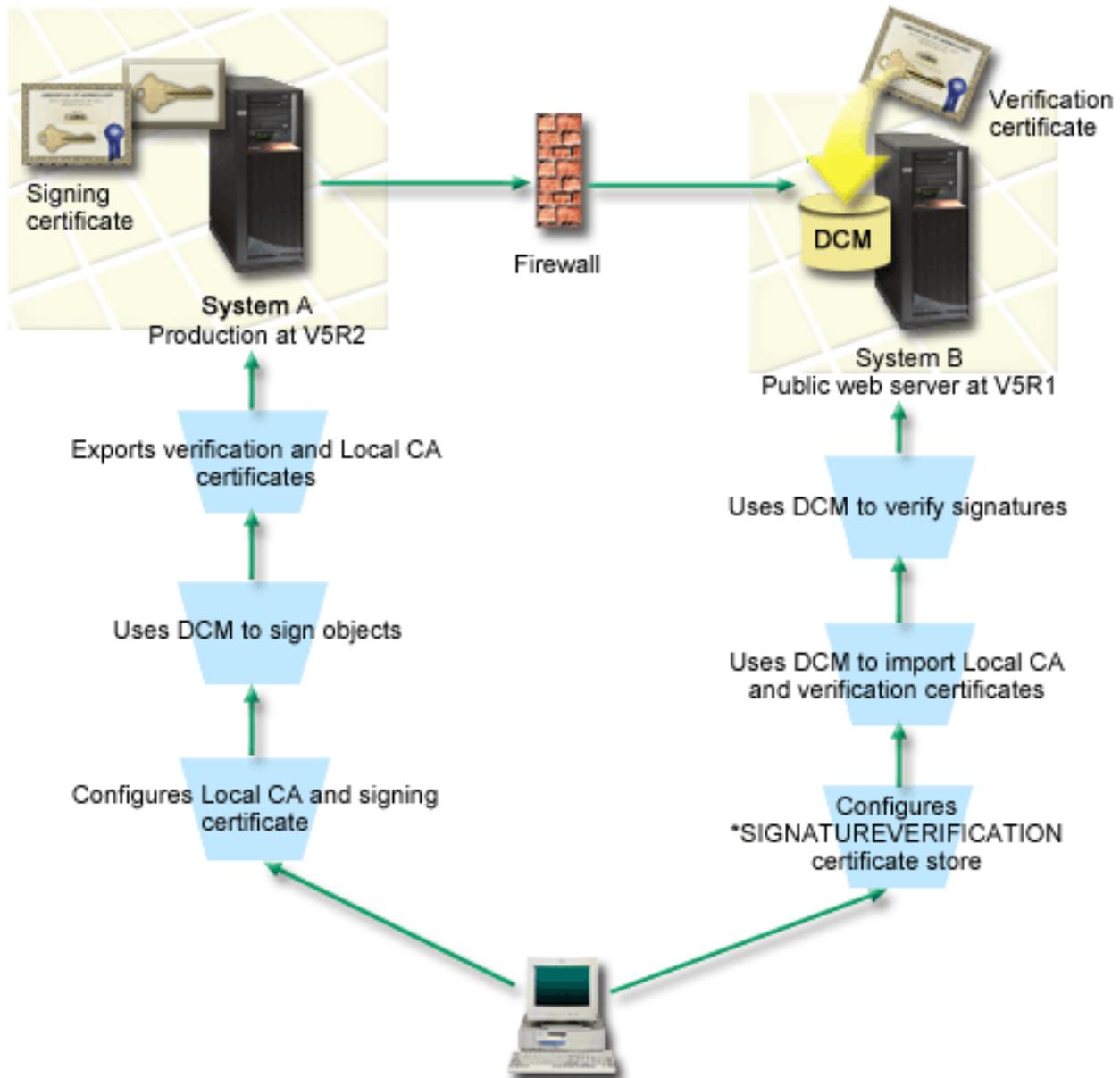
The objectives for this scenario are as follows:

- Company applications and other vulnerable objects on the public Web server (System B) must be signed with a certificate from a Local CA to limit the costs of signing applications.
- System administrators and other designated users must be able to easily verify digital signatures on systems to verify the source and authenticity of company signed objects. To accomplish this, each system must have a copy of both the company's signature verification certificate and the Local Certificate Authority (CA) certificate in each server's \*SIGNATUREVERIFICATION certificate store.
- By verifying the signatures on company applications and other objects, administrators and others can detect whether the content of the objects has changed since they were signed.

- The system administrator must use DCM to sign objects; the system administrator and others must be able to use DCM to verify object signatures.

## Details

The following figure illustrates the object signing and signature verification process for implementing this scenario:



The figure illustrates the following points relevant to this scenario:

### System A

- System A is a System i model that runs OS/400 Version 5 Release 2 (V5R2).
- System A is the company's internal production system and development platform for the public System i Web server (System B).

- System A has a Cryptographic Access Provider 128-bit for System i (5722-AC3) installed.
- System A has Digital Certificate Manager (option 34) and the IBM HTTP Server (5722-DG1) installed and configured.
- System A acts as the Local Certificate Authority (CA) and the object signing certificate resides on this system.
- System A uses DCM to sign objects and is the primary object signing system for the company's public applications and other objects.
- System A is configured to enable signature verification.

### System B

- System B is a System i model that runs OS/400 Version 5 Release 1 (V5R1).
- System B is the company's external public Web server outside the company's firewall.
- System B has a Cryptographic Access Provider 128-bit (5722-AC3) installed.
- System B has Digital Certificate Manager (option 34) and the IBM HTTP Server (5722-DG1) installed and configured.
- System B does not operate a Local CA, nor does System B sign objects.
- System B is configured to enable signature verification by using DCM to create the \*SIGNATUREVERIFICATION certificate store and import the needed verification and Local CA certificates.
- DCM is used to verify signatures on objects.

### Prerequisites and assumptions

This scenario depends on the following prerequisites and assumptions:

1. All systems meet the requirements for installing and using Digital Certificate Manager (DCM).
2. No one has previously configured or used DCM on any of the systems.
3. All systems have the highest level of Cryptographic Access Provider 128-bit licensed program (5722-AC3) installed.
4. The default setting for the verify object signatures during restore (QVfyOBRST) system value on all scenario systems is 3 and has not been changed from this setting. The default setting ensures that the system can verify object signatures as you restore the signed objects.
5. The system administrator for System A must have \*ALLOBJ special authority to sign objects, or the user profile must be authorized to the object signing application.
6. The system administrator or anyone else who creates a certificate store in DCM must have \*SECADM and \*ALLOBJ special authorities.
7. The system administrator or others on all other systems must have \*AUDIT special authority to verify object signatures.

### Configuration task steps

There are two sets of tasks that you must complete to implement this scenario: One set of tasks allows you to configure System A as a Local Certificate Authority (CA) and to sign and verify object signatures. The second set of tasks allows you to configure System B to verify object signatures that System A creates.

See the scenarios details topic presented below to complete these steps.

#### System A task steps

You must complete each of these tasks on System A to create a private Local CA and to sign objects and verify the object signature as this scenario describes:

1. Complete all prerequisite steps to install and configure all needed System i products
2. Use DCM to create a Local Certificate Authority (CA) to issue an object signing certificate.
3. Use DCM to create an application definition
4. Use DCM to assign a certificate to the object signing application definition
5. Use DCM to sign the cgi-bin program objects
6. Use DCM to export the certificates that other systems must use for verifying object signatures You must export both a copy of the Local CA certificate and a copy of the object signing certificate as a signature verification certificate to a file.
7. Transfer the certificate files to the company's public server (System B) so that you and others can verify the signatures that System A creates

### System B task steps

If you intend to restore the signed objects that you transfer to the public Web server in this scenario (System B), you need to complete these signature verification configuration tasks on System B before you transfer the signed objects. Signature verification configuration must be completed before you can successfully verify signatures as you restore the signed objects on the public Web server.

On System B, you must complete these tasks to verify signatures on objects as this scenario describes:

1. Use Digital Certificate Manager (DCM) to create the \*SIGNATUREVERIFICATION certificate store
2. Use DCM to import the Local CA certificate and the signature verification certificate
3. Use DCM to verify the signatures on transferred objects

#### Related information

Digital Certificate Manager (DCM)

### Scenario details: Using DCM to sign objects and verify signatures

Complete the following task steps to configure and use Digital Certificate Manager to sign i5/OS objects as this scenario describes.

#### Step 1: Complete all prerequisite steps

You must complete all prerequisite tasks to install and configure all needed System i products before you can perform specific configuration tasks for implementing this scenario.

#### Step 2: Create a Local Certificate Authority to issue a private object signing certificate

When you use Digital Certificate Manager (DCM) to create a Local Certificate Authority (CA), the process requires you to complete a series of forms. These forms guide you through the process of creating a CA and completing other tasks needed to begin using digital certificates for Secure Sockets Layer (SSL), object signing, and signature verification. Although in this scenario you do not need to configure certificates for SSL, you must complete all forms in the task to configure the system to sign objects.

To use DCM to create and operate a Local CA, follow these steps: Now that you have created a Local CA and an object signing certificate, you must define an object signing application to use the certificate before you can sign objects.

1. Start DCM. Refer to Starting DCM .
2. In the navigation frame of DCM, select **Create a Certificate Authority (CA)** to display a series of forms.

**Note:** If you have questions about how to complete a specific form in this guided task, select the question mark (?) button at the top of the page to access the online help.

3. Complete all the forms for this guided task. As you perform this task, you must do the following:

- a. Provide identifying information for the Local CA.
- b. Install the Local CA certificate in your browser so that your software can recognize the Local CA and validate certificates that the Local CA issues.
- c. Specify the policy data for your Local CA.
- d. Use the new Local CA to issue a server or client certificate that your applications can use for SSL connections.

**Note:** Although this scenario does not make use of this certificate, you must create it before you can use the Local CA to issue the object signing certificate that you need. If you cancel the task without creating this certificate, you must create your object signing certificate and the \*OBJECTSIGNING certificate store in which it is stored separately.

- e. Select the applications that can use the server or client certificate for SSL connections.

**Note:** For the purposes of this scenario, do not select any applications and click **Continue** to display the next form.

- f. Use the new Local CA to issue an object signing certificate that applications can use to digitally sign objects. This subtask creates the \*OBJECTSIGNING certificate store. This is the certificate store that you use to manage object signing certificates.
- g. Select the applications that are to trust your Local CA.

**Note:** For the purposes of this scenario, do not select any applications and click **Continue** to finish the task.

### Step 3: Create an object signing application definition

After you create your object signing certificate, you must use Digital Certificate Manager (DCM) to define an object signing application that you can use to sign objects. The application definition does not need to refer to an actual application; the application definition that you create can describe the type or group of objects that you intend to sign. You need the definition so that you can have an application ID to associate with the certificate to enable the signing process.

To use DCM to create an object signing application definition, follow these steps:

1. In the navigation frame, click **Select a Certificate Store** and select \*OBJECTSIGNING as the certificate store to open.
2. When the Certificate Store and Password page displays, provide the password that you specified for the certificate store when you created it and click **Continue**.
3. In the navigation frame, select **Manage Applications** to display a list of tasks.
4. Select **Add application** from the task list to display a form for defining the application.
5. Complete the form and click **Add**.

Now you must assign your object signing certificate to the application that you created.

### Step 4: Assign a certificate to the object signing application definition

To assign the certificate to your object signing application, follow these steps:

1. In the DCM navigation frame, select **Manage Certificates** to display a list of tasks.
2. From the list of tasks, select **Assign certificate** to display a list of certificates for the current certificate store.
3. Select a certificate from the list and click **Assign to Applications** to display a list of application definitions for the current certificate store.
4. Select one or more applications from the list and click **Continue**. A message page displays to either confirm the certificate assignment or provide error information if a problem occurred.

When you complete this task, you are ready to use DCM to sign the program objects that the company's public Web server (System B) will use.

### Step 5: Sign program objects

To use DCM to sign the program objects for use on the company's public Web server (System B), follow these steps:

1. In the navigation frame, click **Select a Certificate Store** and select **\*OBJECTSIGNING** as the certificate store to open.
2. Enter the password for the **\*OBJECTSIGNING** certificate store and click **Continue**.
3. After the navigation frame refreshes, select **Manage Signable Objects** to display a list of tasks.
4. From the list of tasks, select **Sign an object** to display a list of application definitions that you can use for signing objects.
5. Select the application that you defined in the previous step and click **Sign an Object**. A form displays that allows you to specify the location of the objects that you want to sign.
6. In the field provided, enter the fully qualified path and file name of the object or directory of objects that you want to sign and click **Continue**. Or, enter a directory location and click **Browse** to view the contents of the directory to select objects for signing.

**Note:** You must start the object name with a leading slash or you may encounter an error. You can also use certain wildcard characters to describe the part of the directory that you want to sign. These wildcard characters are the asterisk (\*), which specifies *any number of characters*, and the question mark (?), which specifies *any single character*. For example, to sign all the objects in a specific directory, you might enter `/mydirectory/*`; to sign all the programs in a specific library, you might enter `/QSYS.LIB/QGPL.LIB/*.PGM`. You can use these wildcards only in the last part of the path name; for example, `/mydirectory*/filename` results in an error message. If you want to use the **Browse** function to see a list of library or directory contents, you must enter the wildcard as part of the path name before clicking **Browse**.

7. Select the processing options that you want to use for signing the selected object or objects and click **Continue**.

**Note:** If you choose to wait for job results, the results file displays directly in your browser. Results for the current job are appended to the end of the results file. Consequently, the file may contain results from any previous jobs, in addition to those of the current job. You can use the date field in the file to determine which lines in the file apply to the current job. The date field is in YYYYMMDD format. The first field in the file can be either the message ID (if an error occurred during processing the object) or the date field (indicating the date on which the job processed).

8. Specify the fully qualified path and file name to use for storing job results for the object signing operation and click **Continue**. Or, enter a directory location and click **Browse** to view the contents of the directory to select a file for storing the job results. A message displays to indicate that the job was submitted to sign objects. To view the job results, see job **QOBJSGNBAT** in the job log.

To ensure that you or others can verify the signatures, you must export the necessary certificates to a file and transfer the certificate file to System B. You must also complete all signature verification configuration tasks on System B before you transfer the signed program objects to System B. Signature verification configuration must be completed before you can successfully verify signatures as you restore the signed objects on System B.

### Step 6: Export certificates to enable signature verification on System B

Signing objects to protect the integrity of the contents requires that you and others have a means of verifying the authenticity of the signature. To verify object signatures on the same system that signs the

objects (System A), you must use DCM to create the \*SIGNATUREVERIFICATION certificate store. This certificate store must contain a copy of both the object signing certificate and a copy of the CA certificate for the CA that issued the signing certificate.

To allow others to verify the signature, you must provide them with a copy of the certificate that signed the object. When you use a Local Certificate Authority (CA) to issue the certificate, you must also provide them with a copy of the Local CA certificate.

To use DCM so that you can verify signatures on the same system that signs the objects (System A in this scenario), follow these steps:

1. In the navigation frame, select **Create New Certificate Store** and select \*SIGNATUREVERIFICATION as the certificate store to create.
2. Select **Yes** to copy existing object signing certificates into the new certificate store as signature verification certificates.
3. Specify a password for the new certificate store and click **Continue** to create the certificate store. Now you can use DCM to verify object signatures on the same system that you use to sign objects.

To use DCM to export a copy of the Local CA certificate and a copy of the object signing certificate as a signature verification certificate so that you can verify object signatures on other systems (System B), follow these steps:

1. In the navigation frame, select **Manage Certificates**, and then select the **Export certificate** task.
2. Select **Certificate Authority (CA)** and click **Continue** to display a list of CA certificates that you can export.
3. Select the Local CA certificate that you created earlier from the list and click **Export**.
4. Specify **File** as your export destination and click **Continue**.
5. Specify a fully qualified path and file name for the exported Local CA certificate and click **Continue** to export the certificate.
6. Click **OK** to exit the Export confirmation page. Now you can export a copy of the object signing certificate.
7. Re-select the **Export certificate** task.
8. Select **Object signing** to display a list of object signing certificates that you can export.
9. Select the appropriate object signing certificate from the list and click **Export**.
10. Select **File, as a signature verification certificate** as your destination and click **Continue**.
11. Specify a fully qualified path and file name for the exported signature verification certificate and click **Continue** to export the certificate.

Now you can transfer these files to the endpoint systems on which you intend to verify signatures that you created with the certificate.

## **Step 7: Transfer certificate files to company public server, System B**

You must transfer the certificate files that you created on System A to System B, the company's public Web server in this scenario before you can configure them to verify the objects that you sign. You can use several different methods to transfer the certification files. For example, you might use File Transfer Protocol (FTP) or Management Central package distribution to transfer the files.

## **Step 8: Signature verification tasks: Create \*SIGNATUREVERIFICATION certificate store**

To verify object signatures on System B (the company's public Web server), System B must have a copy of the corresponding signature verification certificate in the \*SIGNATUREVERIFICATION certificate store. Because you used a certificate issued by a Local to sign the objects, this certificate store must also contain a copy of the Local CA certificate.

To create the \*SIGNATUREVERIFICATION certificate store, follow these steps:

1. Start DCM. Refer to Starting DCM .
2. In the Digital Certificate Manager (DCM) navigation frame, select **Create New Certificate Store** and select \*SIGNATUREVERIFICATION as the certificate store to create.

**Note:** If you have questions about how to complete a specific form while using DCM, select the question mark (?) at the top of the page to access the online help.

3. Specify a password for the new certificate store and click **Continue** to create the certificate store. Now you can import certificates into the store and use them to verify object signatures.

### Step 9: Signature verification tasks: Import certificates

To verify the signature on an object, the \*SIGNATUREVERIFICATION store must contain a copy of the signature verification certificate. If the signing certificate is a private one, this certificate store must also have a copy of the Local Certificate Authority (CA) certificate that issued the signing certificate. In this scenario, both certificates were exported to a file and that file was transferred to each endpoint system.

To import these certificates into the \*SIGNATUREVERIFICATION store, follow these steps: You can now use DCM on System B to verify signatures on objects that you created with the corresponding signing certificate on System A.

1. In the DCM navigation frame, click **Select a Certificate Store** and select \*SIGNATUREVERIFICATION as the certificate store to open.
2. When the Certificate Store and Password page displays, provide the password that you specified for the certificate store when you created it and click **Continue**.
3. After the navigation frame refreshes, select **Manage Certificates** to display a list of tasks.
4. From the task list, select **Import certificate**.
5. Select **Certificate Authority (CA)** as the certificate type and click **Continue**.

**Note:** You must import the Local CA certificate before you import a private signature verification certificate; otherwise, the import process for the signature verification certificate will fail.

6. Specify the fully qualified path and file name for the CA certificate file and click **Continue**. A message displays that either confirms that the import process succeeded or provide error information if the process failed.
7. Re-select the **Import certificate** task.
8. Select **Signature verification** as the certificate type to import and click **Continue**.
9. Specify the fully qualified path and file name for the signature verification certificate file and click **Continue**. A message displays that either confirms that the import process succeeded or provides error information if the process failed.

### Step 10: Signature verification tasks: Verify signature on program objects

To use DCM to verify the signatures on the transferred program objects, follow these steps:

1. In the navigation frame, click **Select a Certificate Store** and select \*SIGNATUREVERIFICATION as the certificate store to open.
2. Enter the password for the \*SIGNATUREVERIFICATION certificate store and click **Continue**.
3. After the navigation frame refreshes, select **Manage Signable Objects** to display a list of tasks.
4. From the list of tasks, select **Verify object signature** to specify the location of the objects for which you want to verify signatures.
5. In the field provided, enter the fully qualified path and file name of the object or directory of objects for which you want to verify signatures and click **Continue**. Or, enter a directory location and click **Browse** to view the contents of the directory to select objects for signature verification.

**Note:** You can also use certain wildcard characters to describe the part of the directory that you want to verify. These wildcard characters are the asterisk (\*), which specifies *any number of characters*, and the question mark (?), which specifies *any single character*. For example, to sign all the objects in a specific directory, you might enter `/mydirectory/*`; to sign all the programs in a specific library, you might enter `/QSYS.LIB/QGPL.LIB/*.PGM`. You can use these wildcards only in the last part of the path name; for example, `/mydirectory*/filename` results in an error message. If you want to use the Browse function to see a list of library or directory contents, you must enter the wildcard as part of the path name before clicking **Browse**.

6. Select the processing options that you want to use for verifying the signature on the selected object or objects and click **Continue**.

**Note:** If you choose to wait for job results, the results file displays directly in your browser. Results for the current job are appended to the end of the results file. Consequently, the file may contain results from any previous jobs, in addition to those of the current job. You can use the date field in the file to determine which lines in the file apply to the current job. The date field is in YYYYMMDD format. The first field in the file can be either the message ID (if an error occurred during processing the object) or the date field (indicating the date on which the job processed).

7. Specify the fully qualified path and file name to use for storing job results for the signature verification operation and click **Continue**. Or, enter a directory location and click **Browse** to view the contents of the directory to select a file for storing the job results. A message displays to indicate that the job was submitted to verify object signatures. To view the job results, see job **QOBSJGNBAT** in the job log.

## Scenario: Using APIs to sign objects and verify object signatures

This scenario describes an application development company that wants to programmatically sign the applications that it sells. They want to be able to assure their customers that the applications came from their company and provide them with a means of detecting unauthorized changes to the applications when installing them. Based on the company's business needs and security goals, this scenario describes how to use the i5/OS Sign Object API and the i5/OS Add Verifier API to sign objects and enable signature verification .

### Situation

Your company (MyCo, Inc.) is a business partner that develops applications for customers. As a software developer for the company, you are responsible for packaging these applications for customer distribution. You currently use programs to package an application. Customers can order a compact disc (CD-ROM) or they can visit your Web site and download the application.

You keep current with industry news, especially security news. Consequently, you know that customers are justifiably concerned about the source and content of the programs that they receive or download. There are times when customers think that they are receiving or downloading a product from a trusted source that turns out not to be the true product source. Sometimes this confusion results in customers who install a different product than the one they expected. Sometimes the installed product turns out to be a malicious program or has been altered and damages the system.

Although these types of problems are not common for customers, you want to assure customers that the applications that they obtain from you are really from your company. You also want to provide customers with a method of checking the integrity of these applications so that they can determine whether they have been altered before they install them.

Based on your research, you have decided that you can use i5/OS object signing capabilities to accomplish your security goals. Digitally signing your applications allows your customers to verify that your company is the legitimate source of the application they receive or download. Because you currently package applications programmatically, you have decided that you can use APIs to easily add object

signing to your existing packaging process. You also decide to use a public certificate to sign objects so that you can make the signature verification process transparent to your customers when they install your product.

As part of the application package you include a copy of the digital certificate that you used to sign the object. When a customer obtains the application package, the customer can use the certificate's public key to verify the signature on the application. This process allows the customer to identify and verify the source of the application, as well as ensure that the contents of the application objects have not been altered since they were signed.

This example serves as a useful introduction to the steps involved in programmatically signing objects for applications that you develop and package for others to use.

## Scenario advantages

This scenario has the following advantages:

- Using APIs to package and sign objects programmatically reduces the amount of time that you must spend to implement this security.
- Using APIs to sign objects as you package them decreases the number of steps that you must perform to sign objects because the signing process is part of the packaging process.
- Signing a package of objects allows you to more easily determine whether objects have been changed after they have been signed. This may reduce some of the troubleshooting that you do in the future to track down application problems for customers.
- Using a certificate from a public well-known Certificate Authority (CA) to sign objects allows you to use the Add Verifier API as part of an exit program in your product installation program. Using this API allows you to add the public certificate that you used to sign the application to your customer's system automatically. This ensures that signature verification is transparent to your customer.

## Objectives

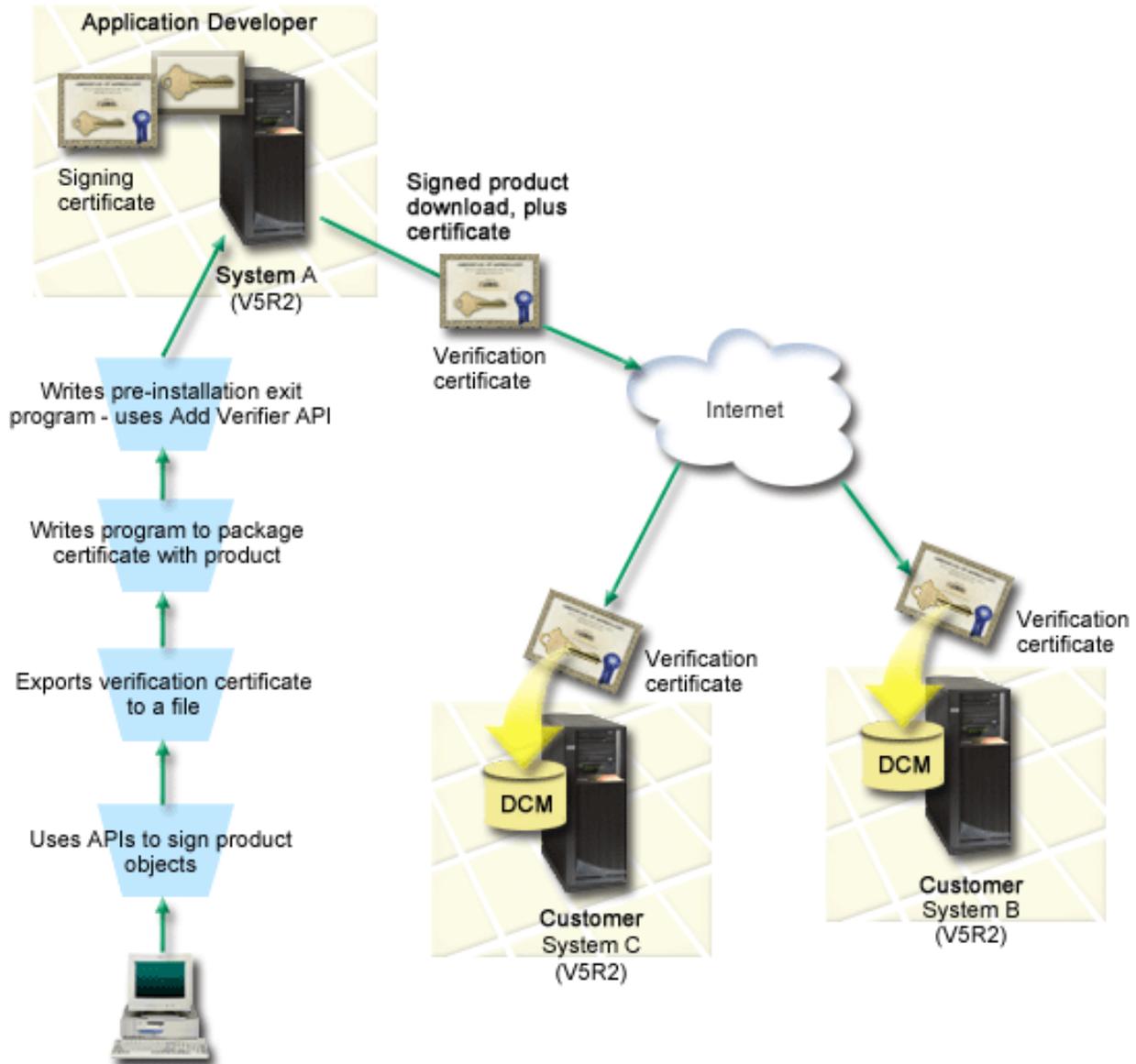
In this scenario, MyCo, Inc. wants to programmatically sign applications that it packages and distributes to their customers. As the application production developer at MyCo, Inc, you currently package your company's applications programmatically for distribution to customers. Consequently, you want to use system APIs to sign your applications and have the customer's system programmatically verify the signature during product installation.

The objectives for this scenario are as follows:

- Company production developer must be able to sign objects by using the Sign Object API as part of an existing programmatic application packaging process.
- Company applications must be signed with a public certificate to ensure that the signature verification process is transparent to the customer during the application product installation process.
- The company must be able to use system APIs to programmatically add the required signature verification certificate to the customer's system \*SIGNATUREVERIFICATION certificate store. The company must be able to programmatically create this certificate store on the customer's system as part of the product installation process if it does not already exist.
- Customers must be able to easily verify digital signatures on the company's application after product installation. Customers must be able to verify the signature so that they can ascertain the source and authenticity of the signed application as well as determine whether changes have been made to the application since it was signed.

## Details

The following figure illustrates the object signing and signature verification process for implementing this scenario:



The figure illustrates the following points relevant to this scenario:

#### Central system A

- System A is a System i model that runs OS/400 Version 5 Release 2 (V5R2).
- System A runs the application developer's product packaging program.
- System A has a Cryptographic Access Provider 128-bit for System i (5722-AC3) installed.
- System A has Digital Certificate Manager (option 34) and the IBM HTTP Server (5722-DG1) installed and configured.
- System A is the primary object signing system for company application products. Product object signing for customer distribution is accomplished on System A by performing these tasks:
  1. Using APIs to sign company application product.
  2. Using DCM to export the signature verification certificate to a file so that customers can verify signed objects.
  3. Writing a program to add the verification certificate to the signed application product.

4. Writing a pre-installation exit program for the product that uses the Add Verifier API. This API allows the product installation process to programmatically add the verification certificate to the \*SIGNATUREVERIFICATION certificate store on the customer's system (Systems B and C).

### Customer systems B and C

- System B is a System i model that runs OS/400® Version 5 Release 2 (V5R2) or a subsequent release of i5/OS.
- System C is a System i model that runs OS/400 Version 5 Release 2 (V5R2) or a subsequent release of i5/OS.
- Systems B and C have Digital Certificate Manager (option 34) and IBM HTTP Server (5722-DG1) installed and configured.
- System B and C purchase and download an application from the Web site of the application development company (which owns System A).
- Systems B and C obtain a copy of the MyCo's signature verification certificate when MyCo's application installation process creates the \*SIGNATUREVERIFICATION certificate store on each of these customer's systems.

### Prerequisites and assumptions

This scenario depends on the following prerequisites and assumptions:

1. All systems meet the requirements for installing and using Digital Certificate Manager (DCM).

**Note:** Meeting the prerequisites for installing and using DCM is an optional requirement for customers (Systems B and C in this scenario). Although the Add Verifier API creates the \*SIGNATUREVERIFICATION certificate store as part of the product install process, if needed, it creates it with a default password. Customers need to use DCM to change the default password to protect this certificate store from unauthorized access.

2. No one has previously configured or used DCM on any of the systems.
3. All systems have the highest level of Cryptographic Access Provider 128-bit licensed program (5722-AC3) installed.
4. The default setting for the verify object signatures during restore (QVfyOBRST) system value on all scenario systems is 3 and has not been changed from this setting. The default setting ensures that the system can verify object signatures as you restore the signed objects.
5. The network administrator for System A must have \*ALLOBJ user profile special authority to sign objects, or the user profile must be authorized to the object signing application.
6. The system administrator or anyone else (including a program) who creates a certificate store in DCM must have \*SECADM and \*ALLOBJ user profile special authorities.
7. System administrators or others on all other systems must have \*AUDIT user profile special authority to verify object signatures.

### Configuration task steps

To sign objects as described in this scenario, refer to the scenario details topic below for steps to complete each of the following tasks on System A :

1. Complete all prerequisite steps to install and configure all needed System i products
2. Use DCM to create a certificate request for obtaining an object signing certificate from a well-known public Certificate Authority (CA)
3. Use DCM to create an object signing application definition
4. Use DCM to import the signed object signing certificate and assign it to your object signing application definition
5. Use DCM to export your object signing certificate as a signature verification certificate so that your customers can use it for verifying the signature on your application objects

6. Update your application packaging program to use the Sign Object API to sign your application
7. Create a pre-installation exit program that uses the Add Verifier API as part of your application packaging process. This exit program allows you to create the \*SIGNATUREVERIFICATION certificate store and add the required signature verification certificate to a customer's system during product installation.
8. Have customers use DCM to reset the default password for the \*SIGNATUREVERIFICATION certificate store on their system

#### **Related information**

Digital Certificate Manager (DCM)

### **Scenario details: Using APIs to sign objects and verify object signatures**

Complete the following task steps to use i5/OS APIs to sign objects as this scenario describes.

#### **Step 1: Complete all prerequisite steps**

You must complete all prerequisite tasks to install and configure all needed System i products before you can perform specific configuration tasks for implementing this scenario.

#### **Step 2: Use DCM to obtain a certificate from a public well-known CA**

This scenario assumes that you have not used Digital Certificate Manager (DCM) previously to create and manage certificates. Consequently, you must create the \*OBJECTSIGNING certificate store as part of the process for creating your object signing certificate. This certificate store, when created, provides the tasks that you need to create and manage object signing certificates. To obtain a certificate from a public well-known Certificate Authority (CA), you use DCM to create the identifying information and the public-private key pair for the certificate and submit this information to the CA to obtain your certificate.

To create the certificate request information that you need to provide to the public well-known CA so that you can obtain your object signing certificate, complete these steps:

1. Start DCM. Refer to Starting DCM .
2. In the navigation frame of DCM, select **Create New Certificate Store** to start the guided task and complete a series of forms. These forms guide you through the process of creating a certificate store and a certificate that you can use to sign objects.

**Note:** If you have questions about how to complete a specific form in this guided task, select the question mark (?) at the top of the page to access the online help.

3. Select **\*OBJECTSIGNING** as the certificate store to create and click **Continue**.
4. Select **Yes** to create a certificate as part of creating the \*OBJECTSIGNING certificate store and click **Continue**.
5. Select **VeriSign or other Internet Certificate Authority (CA)** as the signer of the new certificate, and click **Continue** to display a form that allows you to provide identifying information for the new certificate.
6. Complete the form and click **Continue** to display a confirmation page. This confirmation page displays the certificate request data that you must provide to the public Certificate Authority (CA) that will issue your certificate. The Certificate Signing Request (CSR) data consists of the public key and other information that you specified for the new certificate.
7. Carefully copy and paste the CSR data into the certificate application form, or into a separate file, that the public CA requires for requesting a certificate. You must use all the CSR data, including both the Begin and End New Certificate Request lines. When you exit this page, the data is lost and you cannot recover it.
8. Send the application form or file to the CA that you have chosen to issue and sign your certificate.
9. Wait for the CA to return the signed, completed certificate before you continue to the next task step for the scenario.

### Step 3: Create an object signing application definition

Now that you have sent your certificate request to the well-known public CA, you can use DCM to define an object signing application that you can use to sign objects. The application definition does not need to refer to an actual application; the application definition that you create can describe the type or group of objects that you intend to sign. You need the definition so that you can have an application ID to associate with the certificate to enable the signing process.

To use DCM to create an object signing application definition, follow these steps:

1. In the navigation frame, click **Select a Certificate Store** and select **\*OBJECTSIGNING** as the certificate store to open.
2. When the Certificate Store and Password page displays, provide the password that you specified for the certificate store when you created it and click **Continue**.
3. In the navigation frame, select **Manage Applications** to display a list of tasks.
4. Select **Add application** from the task list to display a form for defining the application.
5. Complete the form and click **Add**.

Once you receive the signed certificate back from the CA, you can assign the certificate to the application that you created.

### Step 4: Import signed public certificate and assign it to the object signing application

To import your certificate and assign it to your application to enable object signing, follow these steps:

1. Start DCM. Refer to Starting DCM .
2. In the navigation frame, click **Select a Certificate Store** and select **\*OBJECTSIGNING** as the certificate store to open.
3. When the Certificate Store and Password page displays, provide the password that you specified for the certificate store when you created it and click **Continue**.
4. After the navigation frame refreshes, select **Manage Certificates** to display a list of tasks.
5. From the task list, select **Import certificate** to begin the process of importing the signed certificate into the certificate store.

**Note:** If you have questions about how to complete a specific form in this guided task, select the question mark (?) at the top of the page to access the online help.

6. Select **Assign certificate** from the **Manage Certificates** task list to display a list of certificates for the current certificate store.
7. Select a certificate from the list and click **Assign to Applications** to display a list of application definitions for the current certificate store.
8. Select your application from the list and click **Continue**. A page displays with either a confirmation message for your assignment selection or an error message if a problem occurred.

When you complete this task, you are ready to sign applications and other objects by using i5/OS APIs. However, to ensure that you or others can verify the signatures, you must export the necessary certificates to a file and transfer them to any system that installs your signed applications. Customer systems must then be able to use the certificate to verify the signature on your application as it installs. You can use the Add Verifier API as part of your application installation program to do the necessary signature verification configuration for your customers. For example, you might create a pre-installation exit program that calls the Add Verifier API to configure your customer's system.

### Step 5: Export certificates to enable signature verification on other systems

Signing objects requires that you and others have a means of verifying the authenticity of the signature and using it to determine whether changes have been made to the signed objects. To verify object

signatures on the same system that signs the objects, you must use DCM to create the \*SIGNATUREVERIFICATION certificate store. This certificate store must contain a copy of both the object signing certificate and a copy of the CA certificate for the CA that issued the signing certificate.

To allow others to verify the signature, you must provide them with a copy of the certificate that signed the object. When you use a Local Certificate Authority (CA) to issue the certificate, you must also provide them with a copy of the Local CA certificate.

To use DCM so that you can verify signatures on the same system that signs the objects (System A in this scenario), follow these steps:

1. In the navigation frame, select **Create New Certificate Store** and select \*SIGNATUREVERIFICATION as the certificate store to create.
2. Select **Yes** to copy existing object signing certificates into the new certificate store as signature verification certificates.
3. Specify a password for the new certificate store and click **Continue** to create the certificate store. Now you can use DCM to verify object signatures on the same system that you use to sign objects.

To use DCM to export a copy of the object signing certificate as a signature verification certificate so that others can verify your object signatures, follow these steps:

1. In the navigation frame, select **Manage Certificates**, and then select the **Export certificate** task.
2. Select **Object signing** to display a list of object signing certificates that you can export.
3. Select the appropriate object signing certificate from the list and click **Export**.
4. Select **File, as a signature verification certificate** as your destination and click **Continue**.
5. Specify a fully qualified path and file name for the exported signature verification certificate and click **Continue** to export the certificate.

Now you can add this file to the application installation package that you create for your product. By using the Add Verifier API as part of your installation program, you can add this certificate to your customer's \*SIGNATUREVERIFICATION certificate store. The API also will create this certificate store if it does not already exist. Your product installation program can then verify the signature on your application objects as it restores them on the customer's systems.

## **Step 6: Update your application packaging program to use system APIs to sign your application**

Now that you have a signature verification certificate file to add to your application package, you can use the Sign Object API to write or edit an existing application to sign your product libraries as you package them for customer distribution.

To help you better understand how to use the Sign Object API as part of your application packaging program, review the following code example. This example code snippet, written in C, is not a complete signing and packaging program; rather it is an example of that portion of such a program that calls the Sign Object API. If you choose to use this program example, change it to suit your specific needs. For security reasons, IBM recommends that you individualize the program example rather than using the default values provided.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 44.

Change this code snippet to fit your needs for using the Sign Object API as part of a packaging program for your application product. You need to pass in two parameters to this program: the name of the library to sign and the name of the object signing application ID; the application ID is case sensitive, the library name is not. The program that you write can call this snippet several times if several libraries are used as part of the product you are signing.

```

/* ----- */
/* */
/* COPYRIGHT (C) IBM CORP. 2004, 2007 */
/* */
/* Use Sign Object API to sign one or more libraries */
/* */
/* The API will digitally sign all objects in a specified library */
/* */
/* */
/* IBM grants you a nonexclusive copyright license to use all */
/* programming code examples from which you can generate similar */
/* function tailored to your own specific needs. */
/* All sample code is provided by IBM for illustrative purposes */
/* only. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these programs. All programs contained herein are */
/* provided to you "AS IS" without any warranties of any kind. */
/* The implied warranties of non-infringement, merchantability and */
/* fitness for a particular purpose are expressly disclaimed. */
/* */
/* */
/* The parameters are: */
/* */
/* char * name of the library to sign */
/* char * name of the application ID */
/* */

#include <qydosgno.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[])
{
    /* parameters:

        char * library to sign objects in,
        char * application identifier to sign with

    */

    int          lib_length, applid_length, path_length, multiobj_length;
    Qus_EC_t     error_code;
    char         libname[11];
    char         path_name[256];

    Qydo_Multi_Objects_T * multi_objects = NULL;
    multiobj_length = 0;
    error_code.Bytes_Provided = 0;    /* return exceptions for any errors */

    /* ----- */
    /* construct path name given library name */
    /* ----- */
    memset(libname, '\00', 11); /* initialize library name */
    for(lib_length = 0;
        ((*argv[1] + lib_length) != ' ') &&
        ((*argv[1] + lib_length) != '\00'));
        lib_length++;
    memcpy(argv[1], libname, lib_length); /* fill in library name */

    /* build path name parm for API call */
    sprintf(path_name, "/QSYS.LIB/%s.LIB/*", libname);

```

```

path_length = strlen(path_name);

/* ----- */
/* find length of application id */
/* ----- */
for(applid_length = 0;
    ((*argv[2] + applid_length) != ' ') &&
    ((*argv[2] + applid_length) != '\00'));
    applid_length++;

/* ----- */
/* sign all objects in this library */
/* ----- */
QYDOSGNO (path_name,          /* path name to object      */
          &path_length,      /* length of path name     */
          "OBJN0100",       /* format name             */
          argv[2],          /* application identifier (ID) */
          &applid_length,   /* length of application ID */
          "1",              /* replace duplicate signature */
          multi_objects,    /* how to handle multiple  */
                          /* objects                  */
          &multiobj_length, /* length of multiple objects */
                          /* structure to use        */
                          /* (0=no mult.object structure)*/
          &error_code);     /* error code              */

return 0;
}

```

## Step 7: Create a pre-installation exit program that uses the Add Verifier API

Now that you have a programmatic process for signing your application, you can use the Add Verifier API as part of your installation program to create your final product for distribution. For example, you might use the Add Verifier API as part of a pre-installation exit program to ensure that the certificate is added to the certificate store before restoring the signed application objects. This allows your installation program to verify the signature on your application objects as they are restored on the customer's system.

**Note:** For security reasons, this API does not allow you to insert a Certificate Authority (CA) certificate into the \*SIGNATUREVERIFICATION certificate store. When you add a CA certificate to the certificate store, the system considers the CA to be a trusted source of certificates. Consequently, the system treats a certificate that the CA issued as having originated from a trusted source. Therefore, you cannot use the API to create an install exit program to insert a CA certificate into the certificate store. You must use Digital Certificate Manager to add a CA certificate to the certificate store to ensure that someone must specifically and manually control which CAs the system trusts. Doing so prevents the possibility that the system might import certificates from sources that an administrator did not knowingly specify as trusted.

If you want to prevent anyone from using this API to add a verification certificate to your \*SIGNATUREVERIFICATION certificate store without your knowledge, you need to consider disabling this API on your system. You can do this by using the system service tools (SST) to disallow changes to security-related system values.

To help you better understand how to use the Add Verifier API as part of your application installation program, review the following pre-installation exit program code example. This example code snippet, written in C, is not a complete pre-installation exit program; rather it is an example of that portion of the program that calls the Add Verifier API. If you choose to use this program example, change it to suit your specific needs. For security reasons, IBM recommends that you individualize the program example rather than using the default values provided.

**Note:** By using the code example, you agree to the terms of the “Code license and disclaimer information” on page 44.

Change this code snippet to fit your needs for using the Add Verifier API as part of a pre-installation exit program to add the required signature verification certificate to your customer’s system as they install your product.

```
/* ----- */
/*
/* COPYRIGHT (C) IBM CORP. 2004, 2007
/*
/* Use Add Verifier API to add a certificate in the specified
/* integrated file system file to the *SIGNATUREVERIFICATION
/* certificate store.
/*
/*
/* The API will create the certificate store if it does not exist.
/* If the certificate store is created it will be given a default
/* password that should be changed using DCM as soon as possible.
/* This warning needs to be given to the owners of the system that
/* use this program.
/*
/*
/*
/* IBM grants you a nonexclusive copyright license to use all
/* programming code examples from which you can generate similiar
/* function tailored to your own specific needs.
/* All sample code is provided by IBM for illustrative purposes
/* only. These examples have not been thoroughly
/* tested under all conditions. IBM, therefore, cannot
/* guarantee or imply reliability, serviceability, or function
/* of these programs. All programs contained herein are
/* provided to you "AS IS" without any warranties of any kind.
/* The implied warranties of non-infringement, merchantability and
/* fitness for a particular purpose are expressly disclaimed.
/*
/*
/*
/* The parameters are:
/*
/* char * path name to integrated file system file that holds
/* the certificate
/* char * certificate label to give certificate
/*
/*
/*
/* ----- */
```

```
#include <qydoadd1.h>
#include <stdlib.h>
#include <string.h>
```

```
int main (int argc, char *argv[])
{
    int          pathname_length, cert_label_length;
    Qus_EC_t     error_code;
    char        * pathname = argv[1];
    char        * certlabel = argv[2];

    /* find length of path name */
    for(pathname_length = 0;
        ((* (pathname + pathname_length) != ' ') &&
         ((* (pathname + pathname_length) != '\000')));
        pathname_length++);

    /* find length of certificate label */
```

```

for(cert_label_length = 0;
    ((*certlabel + cert_label_length) != ' ') &&
    ((*certlabel + cert_label_length) != '\00'));
    cert_label_length++;

error_code.Bytes_Provided = 0;    /* return exceptions for any errors */

QydoAddVerifier (pathname,        /* path name to file with certificate*/
                &pathname_length, /* length of path name           */
                "OBJN0100",      /* format name                   */
                certlabel,        /* certificate label              */
                &cert_label_length, /* length of certificate label    */
                &error_code);    /* error code                     */

return 0;
}

```

With these tasks complete, you can package your application and distribute it to your customers. When they install your application, the signed application objects are verified as part of the installation process. At a later date, customers can use Digital Certificate Manager (DCM) to verify the signature on your application objects. This allows your customers to determine that the source of the application is a trusted one and to determine whether changes have occurred since you signed the application.

**Note:** Your installation program may have created the \*SIGNATUREVERIFICATION certificate store with a default password for your customer. You need to advise your customer that they need to use DCM to reset the password for the certificate store as soon as possible to protect it from unauthorized access.

### Step 8: Have customers reset default password for \*SIGNATUREVERIFICATION certificate store

The Add Verifier API may have created the \*SIGNATUREVERIFICATION certificate store as part of the product install process on your customer's system. If the API created the certificate store, it created a default password for it. Consequently, you need to advise your customers to use DCM to reset this password to protect the certificate store from unauthorized access.

Have your customers complete these steps to reset the \*SIGNATUREVERIFICATION certificate store password:

1. Start DCM. Refer to Starting DCM .
2. In the navigation frame, click **Select a Certificate Store** and select \*SIGNATUREVERIFICATION as the certificate store to open.
3. When the Certificate Store and Password page displays, click **Reset Password** to display the Reset Certificate Store Password page.

**Note:** If you have questions about how to complete a specific form in this guided task, select the question mark (?) at the top of the page to access the online help.

4. Specify a new password for the store, re-enter it to confirm it, select the password expiration policy for the certificate store, and click **Continue**.

### Scenario: Using iSeries Navigator Management Central to sign objects

This scenario describes a company that wants to use i5/OS object signing capabilities to sign objects that it packages and distributes to multiple systems. Based on the company's business needs and security goals, this scenario describes how to use iSeries Navigator's Management Central function to package and sign objects that they distribute to other systems.

## Situation

Your company (MyCo, Inc.) develops applications that it distributes to multiple systems in multiple locations within the company. As the network administrator, you are responsible for ensuring that these applications are installed and updated on all company systems. You currently use iSeries Navigator's Management Central function to more easily package and distribute these applications and to perform other administrative tasks for which you are responsible. However, you spend more time than you like tracking down and correcting problems with these applications because of unauthorized changes to objects. Consequently, you want to better secure the integrity of these objects by digitally signing them.

You have researched i5/OS object signing capabilities and have learned that, beginning in V5R2, Management Central allows you to sign objects when you package and distribute them. By using Management Central you can meet your company's security goals efficiently and relatively easily. You also decide to create a Local Certificate Authority (CA) and use it to issue a certificate to sign objects. Using a certificate issued by a Local CA for object signing limits the expense of using this security technology because you do not need to purchase a certificate from a public well-known CA.

This example serves as a useful introduction to the steps involved in configuring and using object signing for applications that you distribute to multiple company systems.

## Scenario advantages

This scenario has the following advantages:

- Using Management Central to package and sign objects reduces the amount of time that you must spend to distribute signed objects to your company's systems.
- Using Management Central to sign objects in a package decreases the number of steps that you must perform to sign objects because the signing process is part of the packaging process.
- Signing a package of objects allows you to more easily determine whether objects have been changed after they have been signed. This may reduce some of the troubleshooting that you do in the future to track down application problems.
- Using a certificate issued by a Local Certificate Authority (CA) to sign objects makes signing objects less expensive to implement.

## Objectives

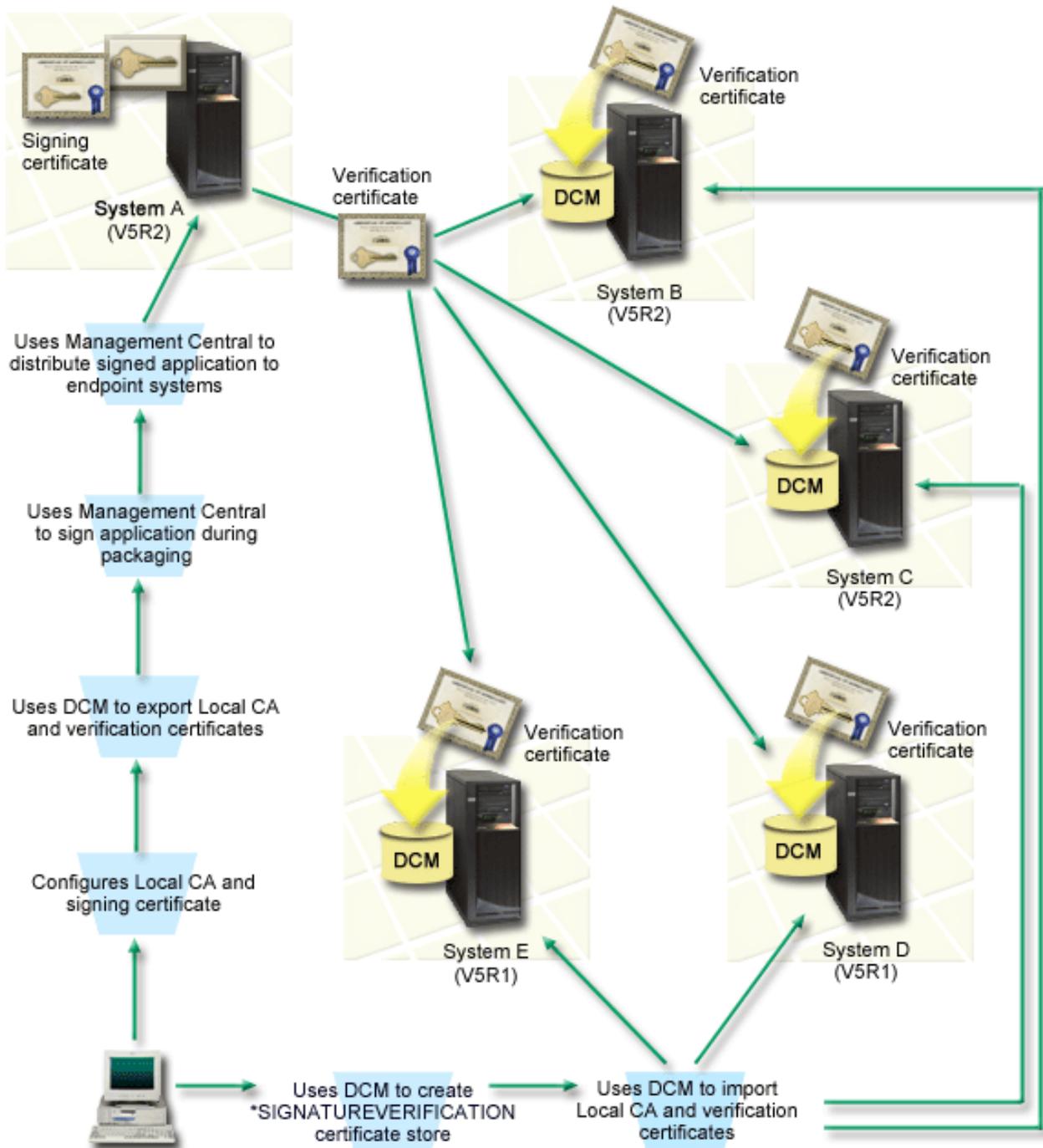
In this scenario, MyCo, Inc. wants to digitally sign applications that it distributes to multiple systems within the company. As the network administrator at MyCo, Inc, you already use Management Central for a number of administrative tasks. Consequently, you want to extend your current use of Management Central to sign the company applications that you distribute to other systems.

The objectives for this scenario are as follows:

- Company applications must be signed with a certificate issued by a Local CA to limit the costs of signing applications.
- System administrators and other designated users must be able to easily verify digital signatures on all systems to verify the source and authenticity of company signed objects. To accomplish this, each system must have a copy of both the company's signature verification certificate and the Local Certificate Authority (CA) certificate in each system's \*SIGNATUREVERIFICATION certificate store.
- Verifying the signatures on company applications allows administrators and others to detect whether the content of the objects has changed since they were signed.
- Administrators must be able to use Management Central to package, sign, and then distribute their applications to their systems.

## Details

The following figure illustrates the object signing and signature verification process for implementing this scenario:



The figure illustrates the following points relevant to this scenario:

### Central system (System A)

- System A is a System i model that runs OS/400 Version 5 Release 2 (V5R2).

- System A serves as the central system from which Management Central functions run, including packaging and distributing company applications.
- System A has a Cryptographic Access Provider 128-bit for System i (5722-AC3) installed.
- System A has Digital Certificate Manager (option 34) and the IBM HTTP Server (5722-DG1) installed and configured.
- System A acts as the Local Certificate Authority (CA) and the object signing certificate resides on this system.
- System A is the primary object signing system for company applications. Product object signing for customer distribution is accomplished on System A by performing these tasks:
  1. Using DCM to create a Local CA and using the Local CA to create an object signing certificate.
  2. Using DCM to export a copy of the Local CA certificate and the signature verification certificate to a file so that endpoint systems (System B, C, D, and E) can verify signed objects.
  3. Using Management Central to sign application objects and package them with the verification certificate files.
  4. Using Management Central to distribute signed application and certificate files to endpoint systems.

### Endpoint systems (Systems B, C, D, and E)

- System B and C are System i models that run OS/400 Version 5 Release 2 (V5R2).
- System D and E are System i models that run OS/400 Version 5 Release 1 (V5R1).
- System B, C, D, and E have Digital Certificate Manager (option 34) and IBM HTTP Server (5722-DG1) installed and configured.
- System B, C, D, and E receive a copy of both the company's signature verification certificate and the Local CA from the central system (System A) when the systems receive the signed application.
- DCM is used to create the \*SIGNATUREVERIFICATION certificate store and import the Local CA and verification certificates into this certificate store.

### Prerequisites and assumptions

This scenario depends on the following prerequisites and assumptions:

1. All systems meet the requirements for installing and using Digital Certificate Manager (DCM).
2. No one has previously configured or used DCM on any of the systems.
3. System A meets the requirements for installing and using iSeries Navigator and Management Central.
4. The Management Central server must be running on all endpoint systems.
5. All systems have the highest level of Cryptographic Access Provider 128-bit licensed program (5722-AC3) installed.
6. The default setting for the verify object signatures during restore (QVIFYOBRST) system value on all scenario systems is 3 and has not been changed from this setting. The default setting ensures that the system can verify object signatures as you restore the signed objects.
7. The network administrator for System A must have \*ALLOBJ user profile special authority to sign objects, or the user profile must be authorized to the object signing application.
8. The network administrator or anyone else who creates a certificate store in DCM must have \*SECADM and \*ALLOBJ user profile special authorities.
9. System administrators or others on all other systems must have \*AUDIT user profile special authority to verify object signatures.

### Configuration task steps

There are two sets of tasks that you must complete to implement this scenario: One set of tasks allows you to set up System A to use Management Central to sign and distribute applications. The other set of

tasks allows system administrators and others to verify the signatures on these applications on all other systems. Refer to the scenario details topic below for the steps to complete these tasks.

### **Object signing task steps**

To sign objects as described in this scenario, refer to the scenario details topic below for steps to complete each of the following tasks on System A :

1. Complete all prerequisite steps to install and configure all needed System i products
2. Use DCM to create a Local Certificate Authority (CA) to issue a private object signing certificate.
3. Use DCM to create an application definition.
4. Use DCM to assign a certificate to the object signing application definition
5. Use DCM to export the certificates that other systems must use for verifying object signatures You must export both a copy of the Local CA certificate and a copy of the object signing certificate as a signature verification certificate to a file.
6. Transfer the certificate files to each endpoint system on which you intend to verify signatures.
7. Use iSeries Navigator Management Central to sign the application objects

### **Signature verification task steps**

You need to complete these signature verification configuration tasks on each endpoint system before you use Management Central to transfer the signed application objects to them. Signature verification configuration must be completed before you can successfully verify signatures as you restore the signed objects on the endpoint systems.

On each endpoint system, you must complete these tasks to verify signatures on objects as this scenario describes:

1. Use DCM to create the \*SIGNATUREVERIFICATION certificate store
2. Use DCM to import the Local CA certificate and the signature verification certificate

#### **Related information**

Digital Certificate Manager (DCM)

### **Scenario details: Using iSeries Navigator Management Central to sign objects**

Complete the following task steps to configure Management Central to sign i5/OS objects as this scenario describes.

#### **Step 1: Complete all prerequisite steps**

You must complete all prerequisite tasks to install and configure all needed System i products before you can perform specific configuration tasks for implementing this scenario.

#### **Step 2: Create a Local Certificate Authority to issue a private object signing certificate**

When you use Digital Certificate Manager (DCM) to create a Local Certificate Authority (CA), the process requires you to complete a series of forms. These forms guide you through the process of creating a CA and completing other tasks needed to begin using digital certificates for Secure Sockets Layer (SSL), object signing, and signature verification. Although in this scenario you do not need to configure certificates for SSL, you must complete all forms in the task to configure the system to sign objects.

To use DCM to create and operate a Local CA, follow these steps: Now that you have created a Local CA and an object signing certificate, you must define an object signing application to use the certificate before you can sign objects.

1. Start DCM. Refer to Starting DCM .

2. In the navigation frame of DCM, select **Create a Certificate Authority (CA)** to display a series of forms.

**Note:** If you have questions about how to complete a specific form in this guided task, select the question mark (?) button at the top of the page to access the online help.

3. Complete all the forms for this guided task. As you perform this task, you must do the following:
  - a. Provide identifying information for the Local CA.
  - b. Install the Local CA certificate in your browser so that your software can recognize the Local CA and validate certificates that the Local CA issues.
  - c. Specify the policy data for your Local CA.
  - d. Use the new Local CA to issue a server or client certificate that your applications can use for SSL connections.

**Note:** Although this scenario does not make use of this certificate, you must create it before you can use the Local CA to issue the object signing certificate that you need. If you cancel the task without creating this certificate, you must create your object signing certificate and the \*OBJECTSIGNING certificate store in which it is stored separately.

- e. Select the applications that can use the server or client certificate for SSL connections.

**Note:** For the purposes of this scenario, do not select any applications and click **Continue** to display the next form.

- f. Use the new Local CA to issue an object signing certificate that applications can use to digitally sign objects. This subtask creates the \*OBJECTSIGNING certificate store. This is the certificate store that you use to manage object signing certificates.
- g. Select the applications that are to trust your Local CA.

**Note:** For the purposes of this scenario, do not select any applications and click **Continue** to finish the task.

### Step 3: Create an object signing application definition

After you create your object signing certificate, you must use Digital Certificate Manager (DCM) to define an object signing application that you can use to sign objects. The application definition does not need to refer to an actual application; the application definition that you create can describe the type or group of objects that you intend to sign. You need the definition so that you can have an application ID to associate with the certificate to enable the signing process.

To use DCM to create an object signing application definition, follow these steps:

1. In the navigation frame, click **Select a Certificate Store** and select \*OBJECTSIGNING as the certificate store to open.
2. When the Certificate Store and Password page displays, provide the password that you specified for the certificate store when you created it and click **Continue**.
3. In the navigation frame, select **Manage Applications** to display a list of tasks.
4. Select **Add application** from the task list to display a form for defining the application.
5. Complete the form and click **Add**.

Now you must assign your object signing certificate to the application that you created.

### Step 4: Assign a certificate to the object signing application definition

To assign the certificate to your object signing application, follow these steps:

1. In the DCM navigation frame, select **Manage Certificates** to display a list of tasks.

2. From the list of tasks, select **Assign certificate** to display a list of certificates for the current certificate store.
3. Select a certificate from the list and click **Assign to Applications** to display a list of application definitions for the current certificate store.
4. Select one or more applications from the list and click **Continue**. A message page displays to either confirm the certificate assignment or provide error information if a problem occurred.

When you complete this task, you are ready to sign objects using Management Central when you package and distribute them. However, to ensure that you or others can verify the signatures, you must export the necessary certificates to a file and transfer them to all the endpoint systems. You must also complete all signature verification configuration tasks on each endpoint system before you use Management Central to transfer the signed application objects to them. Signature verification configuration must be completed before you can successfully verify signatures as you restore the signed objects on the endpoint systems.

### Step 5: Export certificates to enable signature verification on other systems

Signing objects to protect the integrity of the contents requires that you and others have a means of verifying the authenticity of the signature. To verify object signatures on the same system that signs the objects, you must use DCM to create the \*SIGNATUREVERIFICATION certificate store. This certificate store must contain a copy of both the object signing certificate and a copy of the CA certificate for the CA that issued the signing certificate.

To allow others to verify the signature, you must provide them with a copy of the certificate that signed the object. When you use a Local Certificate Authority (CA) to issue the certificate, you must also provide them with a copy of the Local CA certificate.

To use DCM so that you can verify signatures on the same system that signs the objects (System A in this scenario), follow these steps:

1. In the navigation frame, select **Create New Certificate Store** and select **\*SIGNATUREVERIFICATION** as the certificate store to create.
2. Select **Yes** to copy existing object signing certificates into the new certificate store as signature verification certificates.
3. Specify a password for the new certificate store and click **Continue** to create the certificate store. Now you can use DCM to verify object signatures on the same system that you use to sign objects.

To use DCM to export a copy of the Local CA certificate and a copy of the object signing certificate as a signature verification certificate so that you can verify object signatures on other systems, follow these steps:

1. In the navigation frame, select **Manage Certificates**, and then select the **Export certificate** task.
2. Select **Certificate Authority (CA)** and click **Continue** to display a list of CA certificates that you can export.
3. Select the Local CA certificate that you created earlier from the list and click **Export**.
4. Specify **File** as your export destination and click **Continue**.
5. Specify a fully qualified path and file name for the exported Local CA certificate and click **Continue** to export the certificate.
6. Click **OK** to exit the Export confirmation page. Now you can export a copy of the object signing certificate.
7. Re-settle the **Export certificate** task.
8. Select **Object signing** to display a list of object signing certificates that you can export.
9. Select the appropriate object signing certificate from the list and click **Export**.
10. Select **File, as a signature verification certificate** as your destination and click **Continue**.

11. Specify a fully qualified path and file name for the exported signature verification certificate and click **Continue** to export the certificate.

Now you can transfer these files to the endpoint systems on which you intend to verify signatures that you created with the certificate.

### **Step 6: Transfer certificate files to endpoint systems**

You must transfer the certificate files that you created on System A to the endpoint systems in this scenario before you can configure them to verify the objects that you sign. You can use several different methods to transfer the certification files. For example, you might use File Transfer Protocol (FTP) or Management Central package distribution to transfer the files.

### **Step 7: Sign objects by using Management Central**

The object signing process for Management Central is part of the software packaging distribution process. You must complete all signature verification configuration tasks on each endpoint system before you use Management Central to transfer the signed application objects to them. Signature verification configuration must be completed before you can successfully verify signatures as you restore the signed objects on the endpoint systems.

To sign an application that you distribute to endpoint systems as this scenario describes, follow these steps:

1. Use Management Central to package and distribute software products.
2. When you get to the **Identification** panel in the **Product Definition** wizard, click **Advanced** to display the **Advanced Identification** panel.
3. In the **Digital signing** field, enter the application ID for the object signing application that you created earlier and click **OK**.
4. Complete the wizard and continue the process to package and distribute software products with Management Central.

### **Step 8: Signature verification tasks: Create \*SIGNATUREVERIFICATION certificate store on endpoint systems**

To verify object signatures on the endpoint systems in this scenario, each system must have a copy of the corresponding signature verification certificate in the \*SIGNATUREVERIFICATION certificate store. If a private certificate signed the objects, this certificate store must also contain a copy of the Local CA certificate.

To create the \*SIGNATUREVERIFICATION certificate store, follow these steps:

1. Start DCM. Refer to Starting DCM .
2. In the Digital Certificate Manager (DCM) navigation frame, select **Create New Certificate Store** and select **\*SIGNATUREVERIFICATION** as the certificate store to create.

**Note:** If you have questions about how to complete a specific form in this guided task, select the question mark (?) at the top of the page to access the online help.

3. Specify a password for the new certificate store and click **Continue** to create the certificate store. Now you can import certificates into the store and use them to verify object signatures.

### **Step 9: Signature verification tasks: Import certificates**

To verify the signature on an object, the \*SIGNATUREVERIFICATION store must contain a copy of the signature verification certificate. If the signing certificate is a private one, this certificate store must also

have a copy of the Local Certificate Authority (CA) certificate that issued the signing certificate. In this scenario, both certificates were exported to a file and that file was transferred to each endpoint system.

To import these certificates into the \*SIGNATUREVERIFICATION store, follow these steps: Your system can now verify signatures on objects that were created with the corresponding signing certificate when you restore the signed objects.

1. In the DCM navigation frame, click **Select a Certificate Store** and select \*SIGNATUREVERIFICATION as the certificate store to open.
2. When the Certificate Store and Password page displays, provide the password that you specified for the certificate store when you created it and click **Continue**.
3. After the navigation frame refreshes, select **Manage Certificates** to display a list of tasks.
4. From the task list, select **Import certificate**.
5. Select **Certificate Authority (CA)** as the certificate type and click **Continue**.

**Note:** You must import the Local CA certificate before you import a private signature verification certificate; otherwise, the import process for the signature verification certificate will fail.

6. Specify the fully qualified path and file name for the CA certificate file and click **Continue**. A message displays that either confirms that the import process succeeded or provide error information if the process failed.
7. Reselect the **Import certificate** task.
8. Select **Signature verification** as the certificate type to import and click **Continue**.
9. Specify the fully qualified path and file name for the signature verification certificate file and click **Continue**. A message displays that either confirms that the import process succeeded or provides error information if the process failed.

---

## Object signing and signature verification prerequisites

This topic provides information about configuration prerequisites, as well as other planning considerations for signing objects and verifying signatures on your system running the i5/OS operating system.

i5/OS object signing and signature verification capabilities provide you with an additional powerful means of controlling objects on your system. To take advantage of these capabilities, you must meet the prerequisites for using them.

### Object signing prerequisites

There are a number of methods that you can use to sign objects: depending on your business and security needs:

- You can use the Digital Certificate Manager (DCM).
- You can write a program that uses the Sign Object API.
- You can use the Management Central function of iSeries Navigator to sign objects as you package them for distribution to endpoint systems.

Which method you choose for signing objects depends on your business and security needs. Regardless of the method you plan to use to sign objects, you must ensure that certain prerequisite conditions are met:

- You must meet the prerequisites for installing and using Digital Certificate Manager (DCM).
  - You must use DCM to create the \*OBJECTSIGNING certificate store. You create this certificate store either as part of the process of creating a Local Certificate Authority (CA) or as part of the process of managing object signing certificates from a public Internet CA.

- The \*OBJECTSIGNING certificate store must contain at least one certificate, either one that you created by using a Local CA or one that you obtained from a public Internet CA.
- You must use DCM to create at least one object signing application definition to use for signing objects.
- You must use DCM to assign a specific certificate to the object signing application definition.
- The user profile that signs objects must have \*ALLOBJ special authority. The user profile that creates the \*SIGNATUREVERIFICATION certificate store must have \*SECADM and \*ALLOBJ special authorities.

## Signature verification prerequisites

There are a number of methods that you can use to verify signatures on objects:

- You can use the Digital Certificate Manager (DCM).
- You can write a program that uses the Verify Object (QYDOVFYO) API.
- You can use one of a number of commands, such as the Check Object Integrity (CHKOBJITG) command.

Which method you choose for verifying signatures depends on your business and security needs. Regardless of the method you plan to use, you must ensure that certain prerequisite conditions are met:

- You must meet the prerequisites for installing and using Digital Certificate Manager (DCM).
- You must create the \*SIGNATUREVERIFICATION certificate store. You can create this certificate store in one of two ways, depending on your needs. You can create it by using Digital Certificate Manager (DCM) to manage your signature verification certificates. Or, if you are using a public certificate to sign objects, you can create this certificate store by writing a program that uses the Add Verifier (QYDOADDV) API.

**Note:** The Add Verifier API creates the certificate store with a default password. You need to use DCM to reset this default password to one of your choosing to prevent unauthorized access to the certificate store.

- The \*SIGNATUREVERIFICATION certificate store must contain a copy of the certificate that signed the objects. You can add this certificate to the certificate store in one of two ways. You can use DCM on the signing system to export the certificate to a file and then use DCM on the target verification system to import the certificate into the \*SIGNATUREVERIFICATION certificate store. Or, if you are using a public certificate to sign objects, you can add the certificate to the target verification system's certificate store by writing a program that uses the Add Verifier API .
- The \*SIGNATUREVERIFICATION certificate store must contain a copy of the CA certificate that issued the certificate that signed the objects. If you are using a public certificate to sign objects, the certificate store on the target verification system may already have a copy of the required CA certificate. If you are using a certificate issued by a Local CA to sign objects, however, you must use DCM to add a copy of the Local CA certificate to the certificate store on the target verification system.

**Note:** For security reasons, the Add Verifier API does not allow you to insert a Certificate Authority (CA) certificate into the \*SIGNATUREVERIFICATION certificate store. When you add a CA certificate to the certificate store, the system considers the CA to be a trusted source of certificates. Consequently, the system treats a certificate that the CA issued as having originated from a trusted source. Therefore, you cannot use the API to create an install exit program to insert a CA certificate into the certificate store. You must use Digital Certificate Manager to add a CA certificate to the certificate store to ensure that someone must specifically and manually control which CAs the system trusts. Doing so prevents the possibility that the system might import certificates from sources that an administrator did not knowingly specify as trusted.

If you are using a certificate issued by a Local CA to sign objects, you must use DCM on the Local CA host system to export a copy of the Local CA certificate to a file. You can then use

DCM on the target verifying system to import the Local CA certificate into the \*SIGNATUREVERIFICATION certificate store. To prevent a possible error, you must import the Local CA certificate into this certificate store before using the Add Verifier API to add the signature verification certificate. Consequently, if you are using a certificate issued by a Local CA, you may find it easier to use DCM to import both the CA certificate and the verification certificate into the certificate store.

If you want to prevent anyone from using this API to add a verification certificate to your \*SIGNATUREVERIFICATION certificate store without your knowledge, you need to consider disabling this API on your system. You can do this by using the system service tools (SST) to disallow changes to security-related system values.

- The system user profile that verifies signatures must have \*AUDIT special authority. The system user profile that creates the \*SIGNATUREVERIFICATION certificate store or changes the password for it must have \*SECADM and \*ALLOBJ special authorities.

---

## Managing signed objects

Use this information to learn about i5/OS system commands and system values that you can use to work with signed objects and how signed objects affect backup and recovery processes.

Beginning in V5R1, IBM started signing i5/OS. licensed programs, and PTFs as a way of officially marking the operating system as originating from IBM and as a means of detecting when unauthorized changes occur to system objects. Also, business partners and other vendors may be signing the applications that you purchase. Consequently, even if you do not sign objects yourself, you need to understand how to work with signed objects and how these signed objects affect routine system administrative tasks.

Signed objects primarily affect backup and recovery tasks, specifically how you save objects and restore objects onto your system.

## System values and commands that affect signed objects

This topic provides information about i5/OS system values and commands that you can use to manage signed objects or that have an affect on signed objects when you run them.

To manage signed objects effectively, you need to understand how system values and commands affect signed objects. The **Verify object signatures during restore** (QVFYOBJRST) system value determines how certain restore commands affect signed objects and how your system handles signed objects during restore operations. There are no CL commands that are exclusively designed for working with signed objects on a system. However, there are a number of common CL commands that you use to manage signed objects (or to manage the infrastructure objects that make object signing possible). Other commands can adversely affect signed objects on your system by removing the signature from the objects thereby negating the protection that the signature provides.

## System values that affect signed objects

The **Verify object signatures during restore** (QVFYOBJRST) system value, a member of the restore category of i5/OS system values, determines how commands affect signed objects on your system. This system value, which is available through iSeries Navigator, controls how the system handles signature verification during restore operations. The setting that you use for this system value, in conjunction with two other system value settings, affects restore operations for your system. Depending on the setting you select for this value, it can allow or disallow objects from being restored based on their signature status. (For example, whether the object is unsigned, has an invalid signature, is signed by a trusted source, and so forth.) The default setting for this system value allows unsigned objects to be restored, but ensures that signed objects can be restored only if the objects have a valid signature. The system defines an object as

signed only if the object has a signature that your system trusts; the system ignores other, "untrusted" signatures on the object and treats the object as if it is unsigned.

There are several values that you can use for the QVfyOBJRST system value, ranging from ignoring all signatures to requiring valid signatures for all objects that the system restores. This system value only affects executable objects that are being restored, such as programs (\*PGM), commands (\*CMD), service programs (\*SRVPGM), SQL packages (\*SQLPKG), and modules (\*MODULE). It also applies to stream file (\*STMF) objects that have associated Java programs created by Create Java Program (CRTJVAPGM) command. It does not apply to save (\*SAV) files or integrated file system files.

## CL commands that affect signed objects

There are several CL commands that allow you to work with signed objects or that affect signed objects on your system. You can use a variety of commands to view signature information for objects, verify the signature on objects, and save and restore security objects required to verify signatures. Additionally, there are a group of commands that, when run, can remove the signature from objects and negate the security that the signature provides.

### Commands for viewing signature information for an object

- The Display Object Description (DSPOBJD) command. This command shows the names and attributes of specified objects in the specified library or in the libraries of the thread's library list. You can use this command to determine whether an object is signed and to view information about the signature.
- Display Object Links (DSPLNK) and Work with Object Links (WRKLNK) integrated file system commands. You can use either of these commands to display signature information for an object in the integrated file system.

### Commands for verifying object signatures

- Check Object Integrity (CHKOBJITG) command. This command allows you to determine if objects on your system have integrity violations. You can use this command to verify signatures in much the same way that you use a virus checker to determine when a virus has corrupted files or other objects on your system. To learn more about using this command with signed and signable objects, see Code checker commands to ensure signature integrity.
- Check Product Option (CHKPRDOPT) command. This command reports differences between the correct structure and the actual structure of a software product. For example, the command reports an error if an object is deleted from an installed product. You can use the CHKSIG parameter to specify how the command is to handle and report possible signature problems for the product. To learn more about using this command with signed and signable objects, see Code checker commands to ensure signature integrity.
- Save Licensed Program (SAVLICPGM) command. This command saves a copy of the objects that make up a licensed program. It saves the licensed program in a form that can be restored by the Restore Licensed Program (RSTLICPGM) command. You can use the CHKSIG parameter to specify how the command is to handle and report possible signature problems for the product. To learn more about using this command with signed and signable objects, see Code checker commands to ensure signature integrity.
- Restore (RST) command. This command restores a copy of one or more objects that can be used in the integrated file system. This command also allows you restore certificate stores and their contents on the system. However, you cannot use this command to restore the \*SIGNATUREVERIFICATION certificate store. How the restore command handles signed and signable objects is determined by the setting for the Verify object signatures during restore (QVfyOBJRST) system value.
- Restore Library (RSTLIB) command. This command restores one library or a group of libraries that was saved by the Save Library (SAVLIB) command. The RSTLIB command restores the whole library, which includes the library description, object descriptions, and contents of the objects in the library. How this command handles signed and signable objects is determined by the setting for the Verify object signatures during restore (QVfyOBJRST) system value.

- Restore Licensed Program (RSTLICPGM) command. This command loads or restores a licensed program, either for initial installation or new-release installation. How this command handles signed and signable objects is determined by the setting for the Verify object signatures during restore (QVFYOBJRST) system value.
- Restore object (RSTOBJ) command. This command restores one or more objects in a single library, that were saved on diskette, tape, optical volume, or in a save file by using a single command. How this command handles signed and signable objects is determined by the setting for the Verify object signatures during restore (QVFYOBJRST) system value.

## Commands for saving and restoring certificate stores

- Save (SAV) command. This command allows you to save a copy of one or more objects that can be used in the integrated file system, including certificate stores. However, you cannot use this command to save the \*SIGNATUREVERIFICATION certificate store.
- Save Security Data (SAVSECDTA) command. This command allows you to save all security information without requiring the system to be in a restricted state. Using this command allows you to save the \*SIGNATUREVERIFICATION certificate store and the certificates that it contains. This command does not save any other certificate store.
- Save System (SAVSYS) command. This command allows you to save a copy of the licensed internal code and the QSYS library in a format compatible with the installation of the system. It does not save objects from any other library. In addition, it allows you to save the security and configuration objects that you can also save by using the SAVSECDTA and SAVCFG commands. Using this command allows you to save the \*SIGNATUREVERIFICATION certificate store and the certificates that it contains.
- Restore (RST) command. This command allows you restore certificate stores and their contents on the system. However, you cannot use this command to restore the \*SIGNATUREVERIFICATION certificate store.
- Restore User Profiles (RSTUSRPRF) command. This command allows you to restore the basic parts of a user profile or a set of user profiles saved by the Save System (SAVSYS) or the Save Security Data (SAVSECDTA) commands. You can use this command to restore the \*SIGNATUREVERIFICATION certificate store and the stashed passwords for this and all other certificate stores. You can restore the \*SIGNATUREVERIFICATION certificate store without restoring user profile information by specifying \*DCM as the value for the SECDTA parameter and \*NONE for the USRPRF parameter. To use this command to restore user profile information and certificate stores and their passwords, specify \*ALL for the USRPRF parameter.

## Commands that can remove or lose signatures from objects

When you use the following commands on a signed object, you can do so in a manner that might remove or lose the signature from the object. Removing the signature might cause problems with the object affected. At the very least, you will no longer be able to verify the source of the object as a trusted one and will not be able to verify the signature to detect changes to the object. Use these commands only on those signed objects that you have created (as opposed to signed objects that you obtain from others such as IBM or vendors). If you use are concerned that the command removed or lost an object's signature, you can use the Display Object Description (DSPOBJD) command to see if the signature is still there and re-sign it if necessary.

- Note:** To verify whether a Save command lost an object's signature, you must restore the object into a different library than the one from which you saved it (for example, QTEMP). You can then use the DSPOBJD command to determine if the object on the save media lost its signature.
- Change Program (CHGPGM) command. This command changes the attributes of a program without requiring that you recompile it. Also, you can use this command to force re-creation of a program even if the attributes being specified are the same as the current attributes.
  - Change Service Program (CHGSRVPGM) command. This command changes the attributes of a service program without requiring that you recompile it. Also, you can use this command to force re-creation of a service program even if the attributes being specified are the same as the current attributes.

- Clear Save File (CLRSAVF) command. This command clears the contents of a save file; it clears all existing records from the save file and reduces the amount of storage that the file uses.
- Save (SAV) command. This command saves a copy of one or more objects that can be used in the integrated file system. — When using this command, you might lose the signature from command (\*CMD) objects on the save media if you specify a value earlier than V5R2M0 for the TGTRLS parameter. Signature loss occurs because command objects cannot be signed in releases before V5R2.
- Save Library (SAVLIB) command. This command allows you to save a copy of a one or more libraries. When using this command, you might lose the signature from command (\*CMD) objects on the save media if you specify a value earlier than V5R2M0 for the TGTRLS parameter. Signature loss occurs because command objects cannot be signed in releases prior the V5R2.
- Save Object (SAVOBJ) command. This command saves a copy of a single object or a group of objects located in the same library. When using this command, you might lose the signature from command (\*CMD) objects on the save media if you specify a value earlier than V5R2M0 for the TGTRLS parameter. Signature loss occurs because command objects cannot be signed in releases prior the V5R2.

#### Related concepts

“Save and restore considerations for signed objects”

This topic provides information about how signed objects affect how you perform save and restore tasks for your system running the i5/OS operating system.

#### Related information

System value finder

## Save and restore considerations for signed objects

This topic provides information about how signed objects affect how you perform save and restore tasks for your system running the i5/OS operating system.

There are several system values that can affect restore operations for your system. Only one of these system values, the **verify object signatures during restore (QVFYOBJRST)** system value, determines how the system handles signed objects when restoring them. The setting that you choose for this system value lets you determine how the restore process handles verification of objects without signatures or with signatures that are not valid.

Some save and restore commands affect signed objects or determine how your system handles signed and unsigned objects during save and restore operations. You need to be aware of these commands and their affect on signed objects so that you can better manage your system and to avoid potential problems that may occur.

These commands can verify signatures on objects during save and restore operations:

- The Save Licensed Program (SAVLICPGM) command.
- The Restore (RST) command.
- The Restore Library (RSTLIB) command.
- The Restore Licensed Program (RSTLICPGM) command.
- The Restore object (RSTOBJ) command.

These commands allow you to save and restore certificate stores; certificate stores are security-sensitive objects that contain the certificates that you use to sign objects and verify signatures:

- The Save (SAV) command.
- The Save Security Data (SAVSECDTA) command.
- The Save System (SAVSYS) command.
- The Restore (RST) command.
- The Restore User Profiles (RSTUSRPRF) command.

Some save commands, depending on the parameter values that you use, may lose the signature from an object on the save media, thereby negating the security that the signature provides. For example, *any* save operation that refers to a command (\*CMD) object with a target release before V5R2M0 causes the commands to be saved without signatures. Removing the signature might cause problems with the objects affected. At the very least, you will no longer be able to verify the source of the object as a trusted one and will not be able to verify the signature to detect changes to the object. Use these commands only on those signed objects that you have created (as opposed to signed objects that you obtain from others such as IBM or vendors).

**Note:** To verify whether a Save command lost an object's signature, you must restore the object into a different library than the one from which you saved it (for example, QTEMP). You can then use the DSPOBJD command to determine if the object on the save media lost its signature.

You need to be aware of this potential for the following specific save commands, as well as for save commands in general:

- The Save (SAV) command.
- The Save Library (SAVLIB) command.
- The Save Object (SAVOBJ) command.

#### **Related concepts**

“System values and commands that affect signed objects” on page 35

This topic provides information about i5/OS system values and commands that you can use to manage signed objects or that have an affect on signed objects when you run them.

## **Code checker commands to ensure signature integrity**

Learn about using i5/OS commands to verify object signatures to determine object integrity.

You can use Digital Certificate Manager (DCM) or APIs to verify signatures on objects. You can also use several commands to check signatures. Using these commands allows you to verify signatures in much the same way that you use a virus checker to determine when a virus has corrupted files or other objects on your system. Most signatures are checked as the object is restored or installed on to the system, for example by using the RSTLIB command.

You can choose one of three commands to check signatures on objects that are already on the system. Of these, the Check Object Integrity (CHKOBJITG) command is designed specifically for verifying object signatures. Signature checking for each of these commands is controlled by the CHKSIG parameter. This parameter allows you to check all object types that can be signed for signatures, ignore all signatures, or check only objects that have signatures. This last option is the default value for the parameter.

### **Check Object Integrity (CHKOBJITG) command**

The Check Object Integrity (CHKOBJITG) command allows you to determine if objects on your system have integrity violations. You can use this command to check for integrity violations for objects that a specific user profile owns, objects that match a specific path name, or all objects on the system. An integrity violation log entry occurs when one of these conditions is met:

- A command, a program, a module object, or a library's attributes, has been altered.
- The digital signature on an object is determined to be invalid. The signature is an encrypted mathematical summary of the data in the object; therefore, the signature is considered to match and be valid if the data in the object during verification matches the data in the object when it was signed. An invalid signature is determined based on a comparison of the encrypted mathematical summary that is created when the object is signed and the encrypted mathematical summary done during signature verification. The signature verification process compares the two summary values. If the values are not the same, the contents of object have changed since it was signed and the signature is considered to be invalid.

- An object has an incorrect domain attribute for the object type.

If the command detects an integrity violation for an object, it adds the object name, library name (or path name), object type, object owner, and type of failure to a database log file. The command also creates a log entry in certain other cases, although these cases are not integrity violations. For example, the command creates a log entry for objects that are signable but do not have a digital signature, objects that it can not check, and objects in a format that requires changes in order to be used on the current system implementation (IMPI to RISC conversion).

The CHKSIG parameter value controls how the command handles digital signatures on objects. You can specify one of three values for this parameter:

- \*SIGNED – When you specify this value, the command checks objects with digital signatures. The command creates a log entry for any object with a signature that is not valid. This is the default value.
- \*ALL – When you specify this value, the command checks all signable objects to determine whether they have a signature. The command creates a log entry for any signable object that does not have a signature and for any object with a signature that is not valid.
- \*NONE – When you specify this value, the command does not check digital signatures on objects.

### **Check Product Option (CHKPRDOPT) command**

The Check Product Option (CHKPRDOPT) command reports differences between the correct structure and the actual structure of a software product. For example, the command reports an error if an object is deleted from an installed product.

The CHKSIG parameter value controls how the command handles digital signatures on objects. You can specify one of three values for this parameter:

- \*SIGNED – When you specify this value, the command checks objects with digital signatures. The command verifies the signatures on any signed objects. If the command determines that the signature on an object is not valid, the command sends a message to the job log and the identifies the product as being in an erroneous state. This is the default value.
- \*ALL – When you specify this value, the command checks all signable objects to determine whether they have a signature and verifies the signature on these objects. The command sends a message to the job log for any signable object that does not have a signature; however, the command does not identify the product as erroneous. If the command determines that a signature on an object is not valid, it sends a message to the job log and sets the product as erroneous.
- \*NONE – When you specify this value, the command does not check digital signatures on product objects.

### **Save Licensed Program (SAVLICPGM) command**

The Save Licensed Program (SAVLICPGM) command allows you to save a copy of the objects that make up a licensed program. It saves the licensed program in a form that can be restored by the Restore Licensed Program (RSTLICPGM) command.

The CHKSIG parameter value controls how the command handles digital signatures on objects. You can specify one of three values for this parameter:

- \*SIGNED – When you specify this value, the command checks objects with digital signatures. The command verifies the signatures on any signed objects but does not check unsigned objects. If the command determines that the signature on an object is not valid, the command sends a message to the job log to identify the object and the save will fail. This is the default value.
- \*ALL – When you specify this value, the command checks all signable objects to determine whether they have a signature and verifies the signature on these objects. The command sends a message to the

job log for any signable object that does not have a signature; however, the save process does not end. If the command determines that a signature on an object is not valid, it sends a message to the job log and the save will fail.

- \*NONE – When you specify this value, the command does not check digital signatures on product objects.

## Verifying code checker function integrity

Learn how to verify the integrity of the code checker function that you use to verify i5/OS system integrity.

To use the new code checker integrity verification function to that you use to verify the integrity of your system, you must have \*AUDIT special authority.

To verify the code checker function, run the Check System (QydoCheckSystem) API to determine whether any key operating system object has changed since it was signed. When you run the API it checks key system objects, including the programs and service programs and selected command (\*CMD) objects in the QSYS library, as follows:

1. Checks all program (\*PGM) objects to which the system entry point table points.
2. Checks all the service program (\*SRVPGM) objects in the QSYS library and verifies the integrity of the Verify Object API.
3. Runs the Verify Object (QydoVerifyObject) API to verify the integrity of Restore Object (RSTOBJ) command, the Restore Library (RSTLIB) command, and the Check Object Integrity (CHKOBJITG) command.
4. Uses the RSTOBJ and RSTLIB commands on a special save file (\*SAV) to make sure that errors are reporting correctly. A lack of error messages or the wrong error messages indicate a potential problem.
5. Creates a command (\*CMD) object that is designed to fail to verify correctly.
6. Runs the CHKOBJITG command and the Verify Object API on this special command object to ensure that the CHKOBJITG command and the Verify Object API are reporting errors correctly. A lack of error messages or the wrong error messages indicate a potential problem.

### Related concepts

“Code checker integrity verification function” on page 5

This topic provides information about how you can verify the integrity of the code checker function that you use to verify the integrity of your system running the i5/OS operating system.

### Related reference

“Interpreting code checker verification error messages” on page 42

This topic provides information about what messages are returned by the code checker integrity verification function on your system running the i5/OS operating system and how to use these messages to ensure that the code checker function is uncorrupted, as well as possible solutions if messages indicate that the function or key operating system objects may be corrupted.

---

## Troubleshooting signed objects

This topic provides information about i5/OS commands and system values that you can use to work with signed objects and how signed objects affect backup and recovery processes.

When you sign objects and work with signed objects, you may encounter errors that prevent you from accomplishing your tasks and goals. Many of the common errors or problems that you may experience fall into a these categories:

## Troubleshooting object signing errors

This topic provides information about how you can troubleshoot some of the more common problems that you may encounter when signing objects on your system running the i5/OS operating system.

Problem	Possible solution
When using the Sign Object API to sign an object with a target release of V4R5 or earlier, the signing process fails and the object is not signed (error message CPF721).	System does not provide object signing support until V5R1. For those objects that return an error message of CPF721, you must recreate those programs with a target release of V5R1 or later in order to sign them.

## Troubleshooting signature verification errors

This topic provides information about how you can troubleshoot some of the more common problems that you may encounter when verifying i5/OS digital signatures on objects.

Problem	Possible solution
Restore process fails for objects without signatures.	If the lack of a signature is not a concern, check to see if the QVfyOjRST system value is set to 5. A value of 5 specifies that unsigned objects cannot be restored. Change the value to 3 and try the restore again.
Restore process fails for objects with signatures.	This may happen if the *SIGNATUREVERIFICATION certificate store was transferred to the system and DCM was not used to change the password for it. In such a case, the certificates that the store contains cannot be used to verify the signatures on objects during the restore process. Use DCM to change the password for the certificate store. If you do not know the password, you will need to delete the certificate store; recreate it and use DCM to change the password.
When restoring or installing a product, you get an error because a signature failed to verify.	When an object signature fails to verify correctly, the failure may indicate that the object has been changed since it was signed. If object integrity is the issue, do not change the QVfyOjRST system value or perform other actions that might allow the questionable object to restore. Doing so can circumvent the security that signature verification provides and allow a harmful object onto your system. Instead, you need to contact the object signer to determine the appropriate action to take to resolve the problem.

## Interpreting code checker verification error messages

This topic provides information about what messages are returned by the code checker integrity verification function on your system running the i5/OS operating system and how to use these messages to ensure that the code checker function is uncorrupted, as well as possible solutions if messages indicate that the function or key operating system objects may be corrupted.

The following table provides a list of messages that the code checker verification function generates during processing. This table is not a comprehensive list of all messages that you may receive. Instead the table lists those messages most likely to indicate that the code checker verification succeeded fully or that it encountered a serious problem. See the documentation for the Check System (QydoCheckSystem) API for a detailed list of error messages.

Also, a number of messages generated by the code checker verification function as it processes are informational messages and are not listed here. To learn more about how the code checker verification process works, see Verifying code checker function integrity.

Table 1. Code checker verification error messages

Error message	Possible problem and solution
CPFB729	Indicates that the code checker verification process failed to complete as expected. This failure can be caused by a wide range of problems. Review the job log for more detailed error messages to determine the exact nature of the failure and the possible cause. If you determine that key operating system objects failed the integrity check, this failure may indicate that the object has been changed since it was signed when the operating system was shipped. You may need to reinstall the operating system to ensure system integrity.
When reviewing the job log, you see messages such as CPFB723, CPD37A1, or CPD37A0 for these specific objects: <ul style="list-style-type: none"> <li>• Program (*PGM) objects: <ul style="list-style-type: none"> <li>– QYDONOSIG in library QTEMP</li> <li>– QYDOBADSIG in library QTEMP</li> </ul> </li> <li>• Command (*CMD) objects: <ul style="list-style-type: none"> <li>– QYDOBADSIG in library QTEMP</li> <li>– SIGNOFF in library QTEMP</li> </ul> </li> </ul>	Indicates that the special set of objects that the code checker verification function uses for integrity testing failed as expected. This failure indicates that the RSTOBJ command, RSTLIB command, the CHKOBJITG command, and the Verify Object API are reporting errors correctly. No further action is necessary.
CPFB723 for any other object other than those listed previously in this table.	Indicates that the signature on a key operating system object failed to verify. This failure may indicate that the object has been changed since it was signed when the operating system was shipped. You may need to reinstall the operating system to ensure system integrity.
CPFB722 for any other object other than those listed previously in this table.	Indicates that the a key operating system object has no signature when a signature is expected. This lack of signature may indicate that the object has been changed since it was signed when the operating system was shipped. You may need to reinstall the operating system to ensure system integrity.
CPF72A for any other object other than those listed previously in this table.	Indicates that the a key operating system object failed the integrity check. This failure may indicate that the object has been changed since it was signed when the operating system was shipped. You may need to reinstall the operating system to ensure system integrity.

If you ever need to reinstall code that verifies the integrity of the code checker function, you must obtain it from a known, good source. For example, you might load the install media that you used to install the current release. To restore the code checker verification function, follow these steps from an i5/OS command prompt:

1. Run the command QSYS/DLTPGM QSYS/QYDOCHK. This command deletes the Check System (OPM, QYDOCHK; ILE, QydoCheckSystem) API.
2. Run the command QSYS/DLTSRVPGM QSYS/QYDOCHK1. This command deletes the code checker service program with the Check System (OPM, QYDOCHK; ILE, QydoCheckSystem) API.
3. Run the command QSYS/DLTF QSYS/QYDOCHKF. This command deletes the save file that contains the objects that the code checker function uses to test for bad signatures and no signatures
4. Run the command QSYS/RSTOBJ OBJ(QYDOCHK\*) SAVLIB(QSYS) DEV(OPT01) OBJTYPE(\*ALL) OPTFILE('Q5722SS1/Q5200M\_/Q00/Q90'). This command restores all the necessary objects for the code checker verification function from the loaded install media.

**Related tasks**

“Verifying code checker function integrity” on page 41

Learn how to verify the integrity of the code checker function that you use to verify i5/OS system integrity.

---

## Related information for object signing and signature verification

This topic provides links to other resources for learning more about i5/OS object signing and signature verification.

Object signing and signature verification are relatively new security technologies. Here is a small list of other resources that you may find helpful if you are interested in a wider understanding of these technologies and how they work:

- **VeriSign Help Desk Web site**  The VeriSign Web site provides an extensive library on digital certificates topics such as object signing, as well as a number of other Internet security subjects.
- **IBM eServer™ iSeries Wired Network Security: i5/OS V5R1 DCM and Cryptographic Enhancements SG24-6168**  This IBM Redbook focuses on V5R1 network security enhancements. The redbook includes many topics including how to use object signing capabilities, Digital Certificate Manager (DCM), and so forth.

---

## Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

| SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS  
| PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER  
| EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR  
| CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND  
| NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

| UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR  
| ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

- | 1. LOSS OF, OR DAMAGE TO, DATA;
- | 2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC  
| CONSEQUENTIAL DAMAGES; OR
- | 3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

| SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT,  
| INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS  
| OR EXCLUSIONS MAY NOT APPLY TO YOU.

---

## Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Software Interoperability Coordinator, Department YBWA  
3605 Highway 52 N  
Rochester, MN 55901  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

- | The licensed program described in this information and all licensed material available for it are provided
- | by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement,
- | IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

- | SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS
- | PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER
- | EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR
- | CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
- | NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

| UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR  
| ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

- | 1. LOSS OF, OR DAMAGE TO, DATA;
- | 2. SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL  
| DAMAGES; OR
- | 3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

| SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR  
| CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS  
| MAY NOT APPLY TO YOU.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. \_enter the year or years\_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- | eServer
- | i5/OS
- | IBM
- | iSeries
- | OS/400
- | system i
- | xSeries

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft® Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

- | Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

---

## Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

**Personal Use:** You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

**Commercial Use:** You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.





Printed in USA